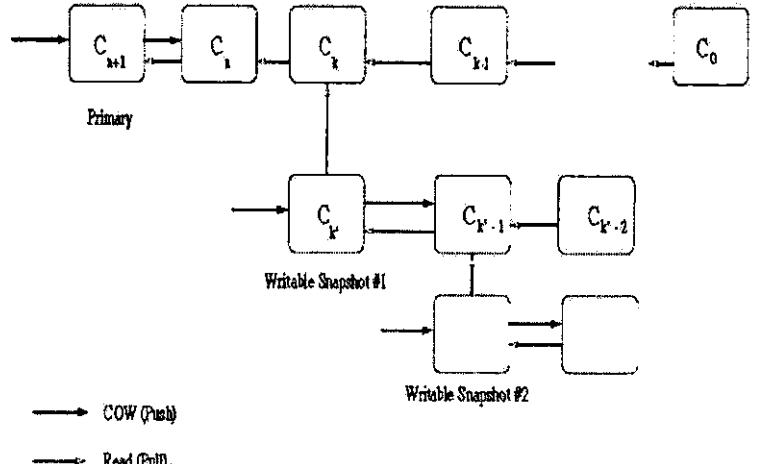
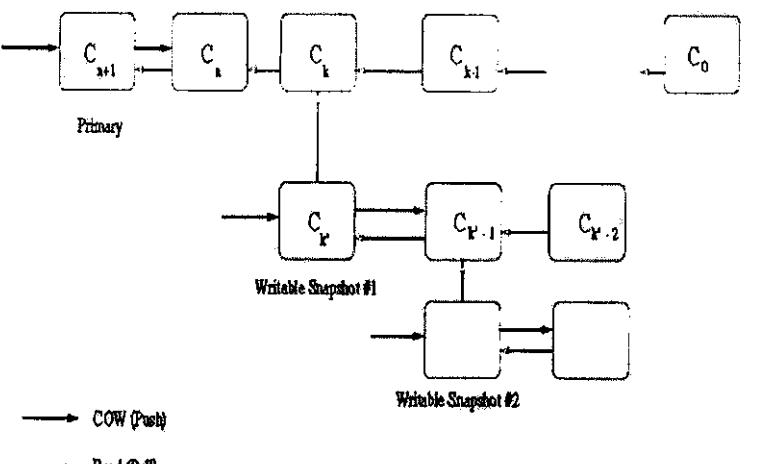
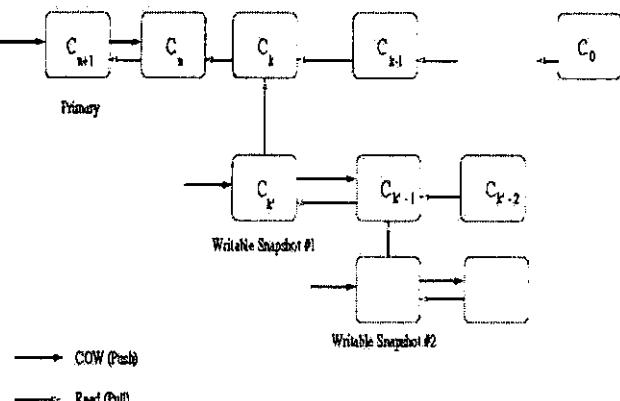


Exhibit E-2

	writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems. Section 3.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 4 shows each snapshot to be a past consistent view of active file systems. Section 3.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.”

	 <p>Figure 4: Snapshot Tree</p>
<p>44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.</p>	<p>Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).</p>
<p>45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new active file system.</p>	<p>Siddha section 3.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.”</p>  <p>Figure 4: Snapshot Tree</p>

<p>46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.</p>	<p>Siddha section 3.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.”</p> 
<p>47. A storage system as in claim 46, wherein snapshot pointers from any of the active file systems to the new active file system are severed.</p>	<p>Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active</p>	<p>It is inherent that the file system in Siddha is executed in conjunction with a computer storage controller. Program and disk implementation of the Siddha file system is disclosed in section 4. One of skill in the art would readily appreciate that the Siddha file system is inherently implemented in conjunction with a computer and/or network interfaces. Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>

file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.

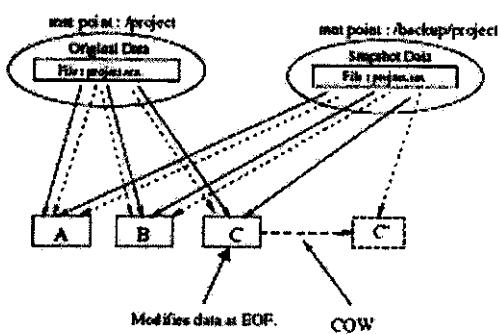


Figure 1: Snapshot COW Mechanism

Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:

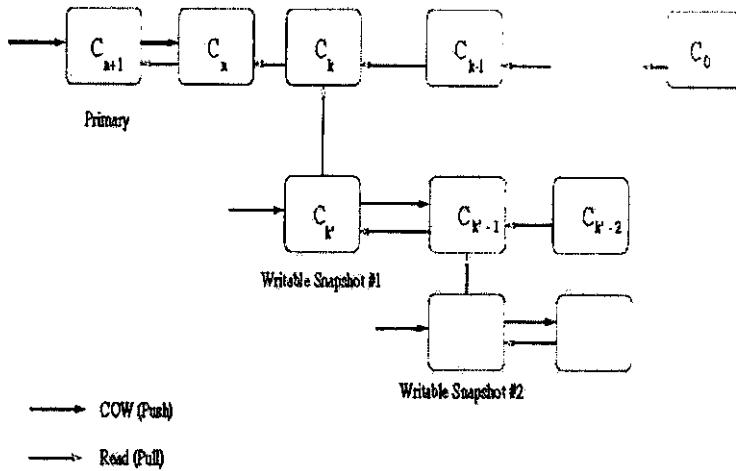


Figure 4: Snapshot Tree

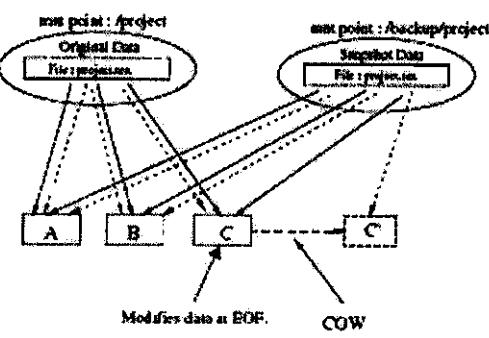
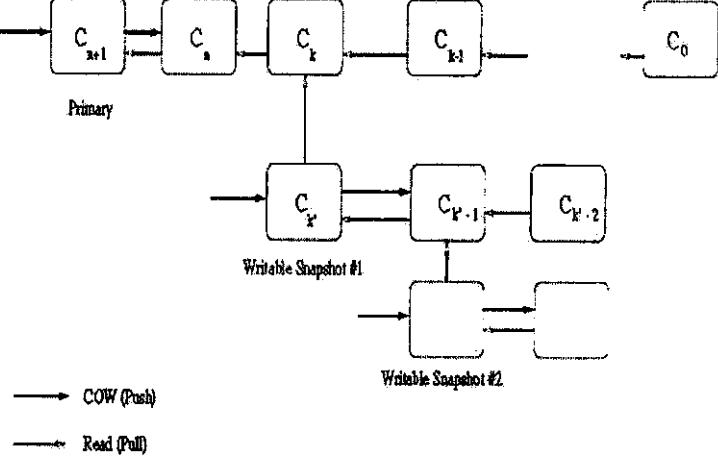
Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot C_{k-1}' , representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.

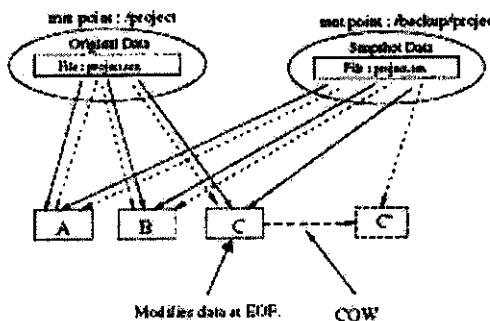
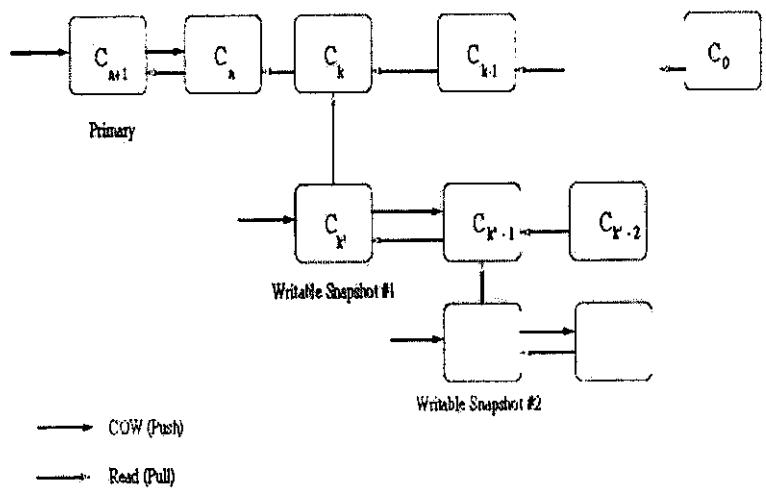
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared

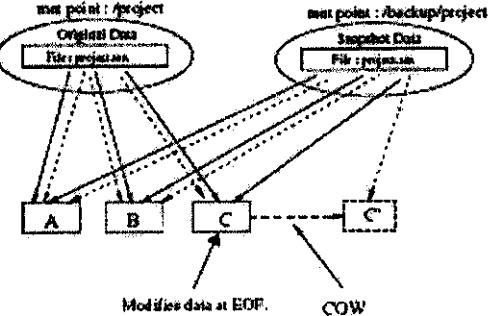
Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.

with the second active file system.	
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	Siddha Fig. 4 teaches that there are no pointers between any of the active file systems. Section 3.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.”
53. A storage system as in claim 52, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Sections 1.1, 3.1.3 and Figs. 1 and 4 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 5 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
54. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially	Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems Section 3.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.” Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as

<p>sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.</p>	<p>writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>55. A storage system as in claim 54, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.</p>	<p>Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.</p>
<p>56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.</p>	<p>Siddha in section 3.1.3 and Fig. 4 teaches snapshots of plural active file systems. Section 3.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.” Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 4 and taught in section 3.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>57. A storage system as in claim 56, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in</p>	<p>Due to the copy-on-write nature of snapshots disclosed in Siddha, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.</p>

<p>a location that is not shared with the third active file system.</p>	
<p>58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>  <p>The diagram illustrates the Snapshot COW Mechanism. It shows two snapshots: 'start point : /project' (Original Data) and 'start point : /backup/project' (Snapshot Data). The 'Original Data' is represented by a box containing 'File 1: project.txt'. The 'Snapshot Data' is represented by a box containing 'File 1: project.txt'. Below these are three boxes labeled A, B, and C. Box A is connected to both snapshots. Box B is connected to the 'Original Data' snapshot. Box C is connected to the 'Snapshot Data' snapshot. A dashed arrow labeled 'Modifies data at EOF.' points from Box C to Box B. A solid arrow labeled 'COW' points from Box C to Box A.</p> <p>Figure 1: Snapshot COW Mechanism</p> <p>Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:</p>  <p>The diagram illustrates a Snapshot Tree. It shows a primary sequence of file systems: $C_{k+1} \rightarrow C_k \rightarrow C_{k-1} \rightarrow C_{k-2} \rightarrow \dots \rightarrow C_0$. A 'Writable Snapshot #1' is shown branching off from C_k to $C_{k,1} \rightarrow C_{k,2}$. A 'Writable Snapshot #2' is shown branching off from $C_{k,1}$ to an unnamed node. A legend at the bottom indicates: a solid arrow pointing right labeled 'COW (Push)' and a dashed arrow pointing right labeled 'Read (Pull)'.</p> <p>Figure 4: Snapshot Tree</p> <p>Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot $C_{k'}$ initially shares data with snapshot C_k, which represents the state of the original file system at time k. Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1}, which represents the state of the original</p>

	file system.
59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	<p>Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>  <p>Figure 1: Snapshot COW Mechanism</p> <p>Siddha section 3.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 4:</p>  <p>Figure 4: Snapshot Tree</p> <p>Siddha teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 4, writable snapshot $C_{k'}$ initially shares data with snapshot C_k, which represents the state of the original file system at time k. Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1}, which represents the state of the original file system.</p>

<p>60. A method of operating data storage, comprising: making a snapshot of organizational data of a first active file system, the snapshot pointing to original non-organizational data of the first active file system; storing the snapshot; modifying a first portion of the original non-organizational data of the first active file system in response to a first active file system access request, resulting in a modified first portion being part of first modified non-organizational data of the first active file system; and storing the modified first portion so as not to overwrite the first portion; wherein, after the step of storing the modified first portion, the snapshot points to the original non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data of the first filing system, and the original non-organizational data and the first modified non-organizational data partially overlap.</p>	<p>Siddha generally and section 3.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>  <p>Figure 1: Snapshot COW Mechanism</p> <p>As Fig. 1 shows that the snapshot originally points to data blocks A-C. When data block C is modified, the snapshot points to the original block, while the active file system points to the modified data block C'. The active file system and snapshot data sets partially overlap.</p>
<p>61. A method according to claim 60, wherein: the step of storing the snapshot comprises storing the snapshot as a second active filing system; the method further comprising: modifying a second portion</p>	<p>Siddha section 3.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 4.”</p>

of the original non-organizational data in response to a second active file system access request, resulting in a modified second portion being part of second modified non-organizational data of the second active file system; and storing the modified second portion so as not to overwrite the second portion; wherein, after the step of storing the modified second portion, the snapshot points to the second modified non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data, and the first modified non-organizational data and the second modified non-organizational data partially overlap.

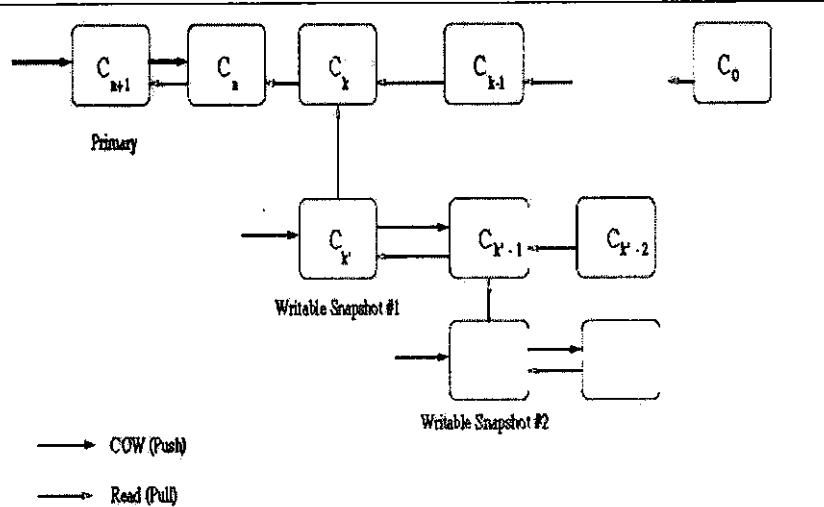


Figure 4: Snapshot Tree

In Fig. 4, snapshot C_k' has been mounted as writable and originally shares data with C_k . As the two active file systems diverge over time, their data sets will partially overlap in the manner shown in Fig. 1.

62. A method according to claim 61, wherein the step of making a snapshot is performed at a consistency point of the first active file system.

Siddha sections 1.1 and 3.1.1 teach that snapshots are taken at a consistency point. Fig 4. further shows that writable snapshot C_k' is mounted at a consistency point C_k of the original active file system.

63. A method according to claim 62, wherein data is stored in the first and second active file systems using blocks.

Siddha section 1.1 and Fig. 1 teach that data is stored using blocks.

Fourth Basis of Invalidity

The reference applicable to the fourth basis of invalidity is:

1. Sun StorEdge Instant Image 2.0 System Administrator's Guide, February 2000 ("Sun").

The pertinence and manner of applying Sun to claims 1-59 for which re-examination is requested is as follows:

1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks.
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a

respective active file system at a past consistency point.	snapshot.
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
7. A method as in claim 5, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
8. A method as in claim 7, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
9. A method as in claim 8, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”

11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
13. A method as in claim 10, further comprising the step of severing any snapshot pointers from the first active file system to the second active file system.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
14. A method as in claim 10, further comprising the steps of making snapshots of ones of the plural active file systems.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
16. A method as in claim 10, further comprising the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a

	synchronization.
17. A method as in claim 16, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
18. A method as in claim 10, further comprising the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
19. A method as in claim 18, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the

	shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks.
22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
25. A method as in claim 24, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
26. A memory as in claim 24, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.

27. A memory as in claim 26, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
28. A memory as in claim 27, wherein snapshot pointers from any of the active file systems to the new active file system are severed.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.

32. A memory as in claim 29, wherein the instructions further comprise the step of severing any snapshot pointers from the first active file system to the second active file system.	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
34. A memory as in claim 33, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
35. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
36. A memory as in claim 35, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
37. A memory as in claim 29, wherein the instructions further comprise the steps of:	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-

<p>making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.</p>	<p>time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.</p>
<p>38. A memory as in claim 37, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.</p>	<p>Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.</p>
<p>39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. It is inherent that Sun Instant Image 2.0 software can be used in conjunction with a storage controller and a computer or network. Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A</p>

	bitmap volume file tracks the differences between the two.”
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Sun teaches at pp. 1-5 through 1-6 that the dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks.
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new active file system.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.	Sun teaches at pp. 1-5 through 1-6 that once mounted, a shadow volume can be read and written to, becoming an active file system.
47. A storage system as in claim 46, wherein snapshot pointers from any of the active file	Because master and shadow volumes can diverge over time, they are independent and

systems to the new active file system are severed.	inherently do not share pointers.
48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	<p>Sun Instant Image 2.0 software is in the form of processor-executable instructions stored in a computer memory. It is inherent that Sun Instant Image 2.0 software can be used in conjunction with a storage controller and a computer or network.</p> <p>Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”</p>
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first	Because master and shadow volumes can diverge over time, they are independent and inherently do not share pointers.

active file system to the second active file system.	
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot.
53. A storage system as in claim 52, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Each shadow volume inherently includes a complete hierarchy of data that is a copy of the data on the master volume at a point in time. This includes file system data when a file system is mounted on the volumes. As the master and shadow volumes diverge over time, any changes in file system data will be unique to each volume.
54. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.
55. A storage system as in claim 54, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.
56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active	Once the shadow volume is mounted, it can be treated as a master and have its own point-in-time shadow volumes, equivalent to a snapshot. Thus, the shadow volume feature in Sun can be used to create multiple active volumes.

<p>file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.</p>	<p>Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.” Thus, changes to each of the writable volumes are not reflected in the other writable volumes absent a synchronization.</p>
<p>57. A storage system as in claim 56, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.</p>	<p>Because master and shadow volume data sets can diverge over time, it is inherent that modified data is recorded in different locations so as to not overwrite any other data.</p>
<p>58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”</p>
<p>59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes</p>	<p>Sun teaches at pp. 1-5 through 1-6 a master volume (volume containing the original data) and a shadow volume (volume that is the copy of the master volume). The shadow volume can be independent (produced via a full volume copy operation) or dependent. The dependent shadow volume is a point-in-time</p>

made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	copy that relies on the master for all unmodified data blocks. When a data block in the master volume is modified, a copy is placed in the dependent shadow, so as to preserve point-in-time consistency of the shadow block. Thus, dependent shadow volumes use a copy-on-write technique. Sun further teaches at p. 1-6 that “[o]nce the shadow volume is created, you can read from and write to it. After you write to it, the shadow volume data is unique and does not necessarily match the master volume. A bitmap volume file tracks the differences between the two.”
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fifth Basis of Invalidity

The reference applicable to the fifth basis of invalidity is:

1. S. B. Siddha, *Persistent Snapshots*, A Project Report Submitted in partial fulfillment of the requirement for the Degree of Master of Engineering in Computer Science and Engineering, INDIAN INSTITUTE OF SCIENCE, January, 2000. (“Siddha Report”).

The pertinence and manner of applying Siddha Report to claims 1-63 for which re-examination is requested is as follows:

1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Siddha Report generally and section 1.1 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

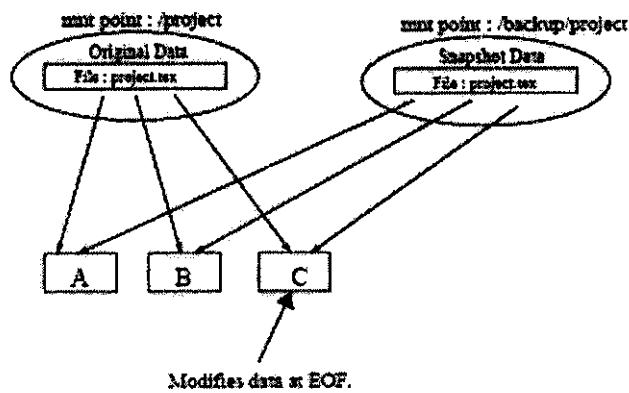


Figure 1.1: Before COW

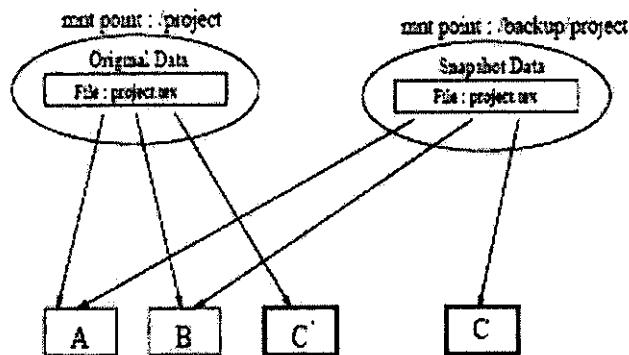


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

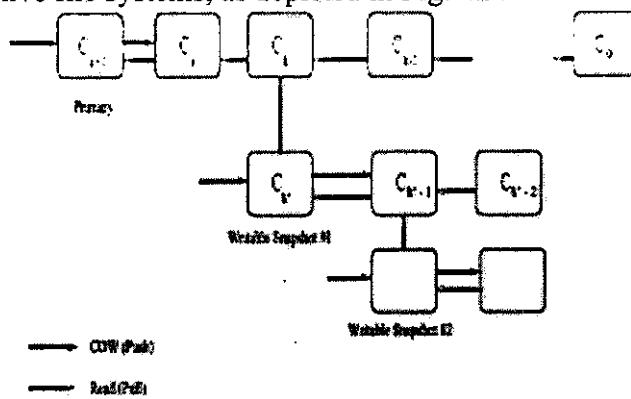


Figure 2.3: Snapshot Tree

Snapshot C₁ is even labeled as "Writable Snapshot #1." Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as

	writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 2.3 shows each snapshot to be a past consistent view of active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”

	<p>The diagram illustrates a Snapshot Tree structure. At the top, a horizontal chain of six boxes represents the primary file system: $C_{p,1}$, C_p, C_1, $C_{1,1}$, $C_{1,2}$, and C_2. Below this, a vertical line connects C_1 to a second horizontal chain of three boxes: C_r, $C_{r,1}$, and $C_{r,2}$. This second chain is labeled "Writable Snapshot 1". Below "Writable Snapshot 1", another vertical line connects $C_{r,1}$ to a third horizontal chain of two boxes: $C_{r,1,1}$ and $C_{r,1,2}$. This third chain is labeled "Writable Snapshot 2". A legend at the bottom indicates that a solid line with an arrowhead pointing right represents "COW Path" and a dashed line with an arrowhead pointing right represents "Read Path".</p> <p>Figure 2.3: Snapshot Tree</p>
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	<p>Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).</p>
7. A method as in claim 5, wherein at least one of the snapshots is converted into a new active file system.	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>The diagram illustrates a Snapshot Tree structure. At the top, a horizontal chain of six boxes represents the primary file system: $C_{p,1}$, C_p, C_1, $C_{1,1}$, $C_{1,2}$, and C_2. Below this, a vertical line connects C_1 to a second horizontal chain of three boxes: C_r, $C_{r,1}$, and $C_{r,2}$. This second chain is labeled "Writable Snapshot 1". Below "Writable Snapshot 1", another vertical line connects $C_{r,1}$ to a third horizontal chain of two boxes: $C_{r,1,1}$ and $C_{r,1,2}$. This third chain is labeled "Writable Snapshot 2". A legend at the bottom indicates that a solid line with an arrowhead pointing right represents "COW Path" and a dashed line with an arrowhead pointing right represents "Read Path".</p> <p>Figure 2.3: Snapshot Tree</p>
8. A method as in claim 7, wherein the one of the snapshots is converted by making the one of the snapshots writable.	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p>

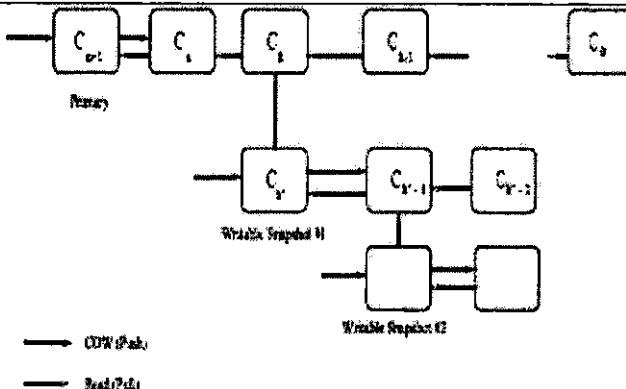


Figure 2.3: Snapshot Tree

<p>9. A method as in claim 8, wherein snapshot pointers from any of the active file systems to the new active file system are severed.</p>	<p>Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>

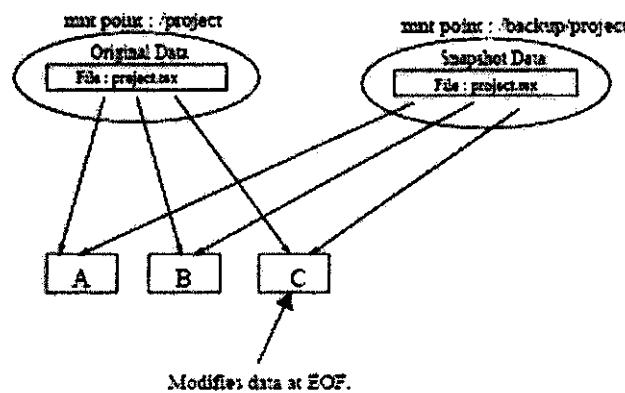


Figure 1.1: Before COW

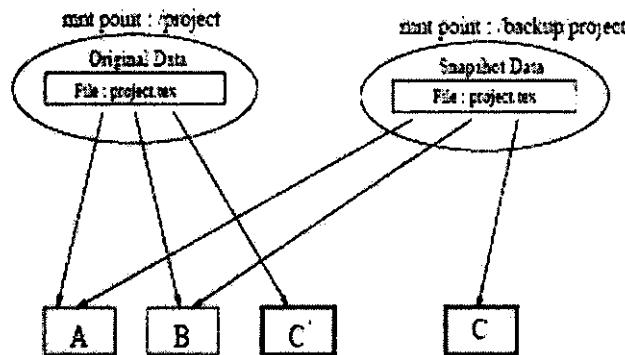


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

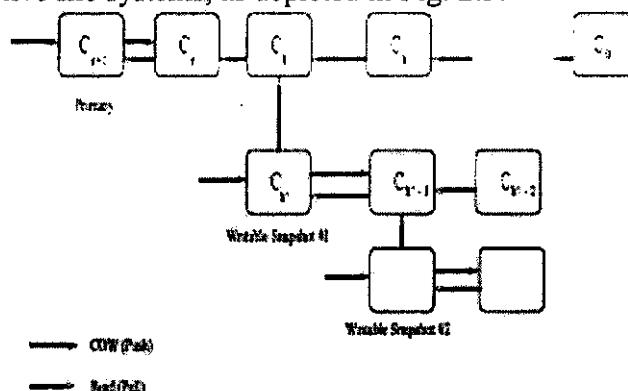


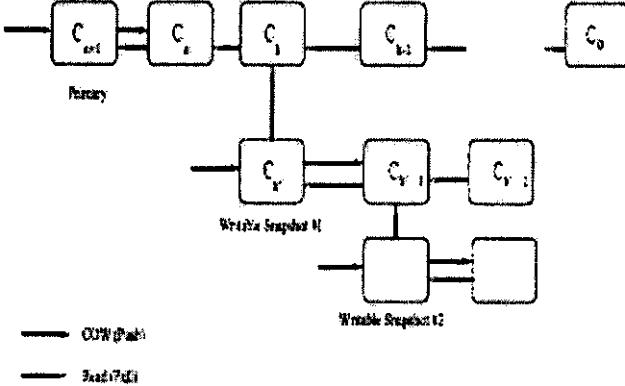
Figure 2.3: Snapshot Tree

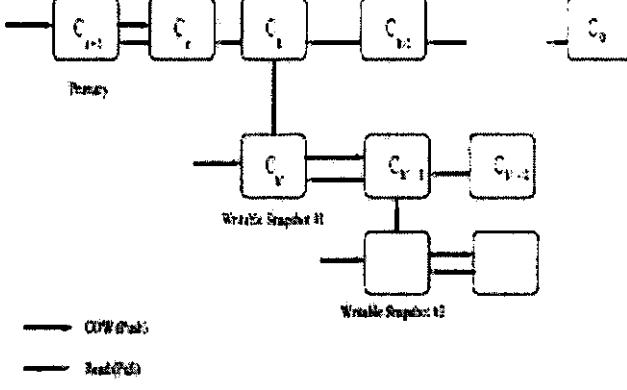
Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable

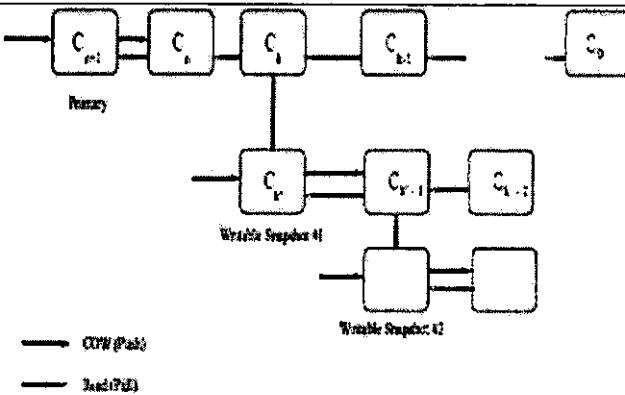
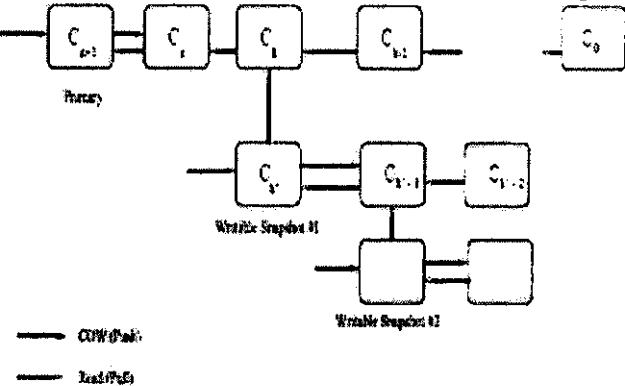
	snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
13. A method as in claim 10, further comprising the step of severing any snapshot pointers from the first active file system to the second active file system.	Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”
14. A method as in claim 10, further comprising the steps of making snapshots of ones of the plural active file systems.	Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file	Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).

systems.	
16. A method as in claim 10, further comprising the steps of: making anew snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in section 2.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
17. A method as in claim 16, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
18. A method as in claim 10, further comprising the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in section 2.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>

<p>19. A method as in claim 18, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.</p>	<p>Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.</p>
<p>20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>It is inherent that the file system in Siddha Report is in the form of computer-executable instructions stored in a memory. Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p> <div style="text-align: center;"> <p>mnt point : /project Original Data File : project.txt</p> <p>mnt point : /backup/project Snapshot Data File : project.txt</p> <p>A B C</p> <p>Modifies data at EOF.</p> </div> <p>Figure 1.1: Before COW</p> <div style="text-align: center;"> <p>mnt point : /project Original Data File : project.txt</p> <p>mnt point : /backup/project Snapshot Data File : project.txt</p> <p>A B C' C</p> </div> <p>Figure 1.2: After COW</p> <p>Siddha Report section 2.1.3 further teaches that snapshots can be</p>

	<p>made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:</p>  <p>Legend:</p> <ul style="list-style-type: none"> — COW Path — Shared Path <p>Figure 2.3: Snapshot Tree</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C_k' initially shares data with snapshot C_k, which represents the state of the original file system at time k. Snapshot $C_k'-1$, representing the state of the new file system at a later time, does not share changed data with snapshot C_k-1, which represents the state of the original file system.</p>
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_k'-1$, representing the state of the new file system at a later time, does not share changed data with snapshot C_k-1 , which represents the state of the original file system.
22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
23. A memory as in claim 21, wherein when changes are made to the second	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as

<p>active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.</p>	<p>shown in Fig. 1.</p>
<p>24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.</p>	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 2.3 shows each snapshot to be a past consistent view of active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> 
<p>25. A method as in claim 24, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.</p>	<p>Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).</p>
<p>26. A memory as in claim 24, wherein at least one of the snapshots is converted into a new active file system.</p>	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p>

	 <p>Figure 2.3: Snapshot Tree</p>
<p>27. A memory as in claim 26, wherein the one of the snapshots is converted by making the one of the snapshots writable.</p>	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p>  <p>Figure 2.3: Snapshot Tree</p>
<p>28. A memory as in claim 27, wherein snapshot pointers from any of the active file systems to the new active file system are severed.</p>	<p>Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the</p>	<p>It is inherent that the file system in Siddha Report is in the form of computer-executable instructions stored in a memory. Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>

snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.

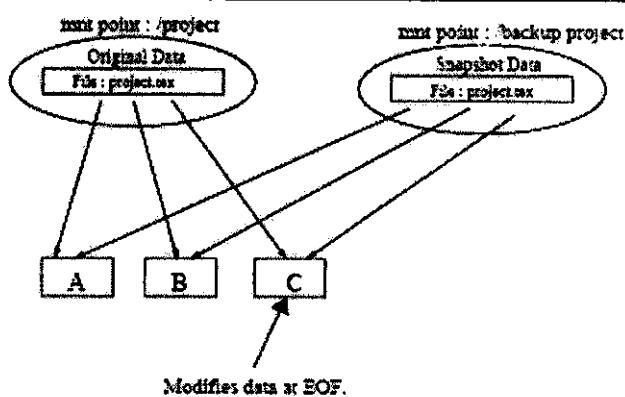


Figure 1.1: Before COW

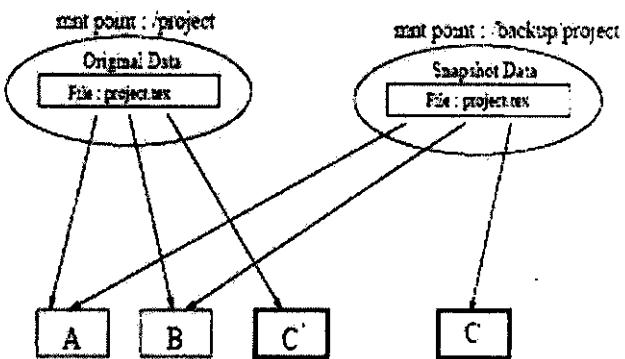


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

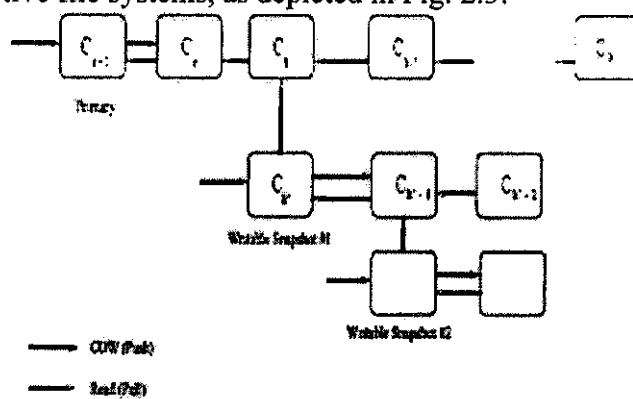


Figure 2.3: Snapshot Tree

Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable

	snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
32. A memory as in claim 29, wherein the instructions further comprise the step of severing any snapshot pointers from the first active file system to the second active file system.	Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”
34. A memory as in claim 33, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for	Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).

the plural active file systems.	
35. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in section 2.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
36. A memory as in claim 35, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	<p>Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.</p>
37. A memory as in claim 29, wherein the instructions further comprise the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in section 2.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot</p>

file system.	references this one."
38. A memory as in claim 37, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	It is inherent that the file system in Siddha Report is executed in conjunction with a computer storage controller. Program and disk implementation of the Siddha Report file system is disclosed in section 4. Thus, the Siddha Report file system is inherently implemented in conjunction with a computer or network interfaces. Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

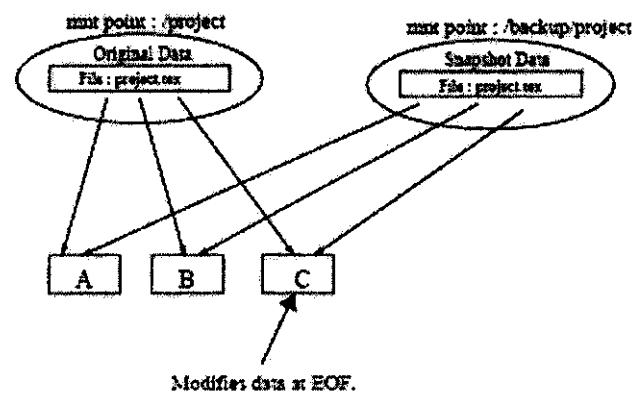


Figure 1.1: Before COW

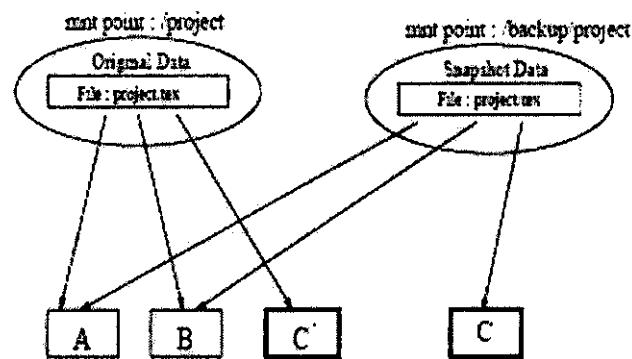


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

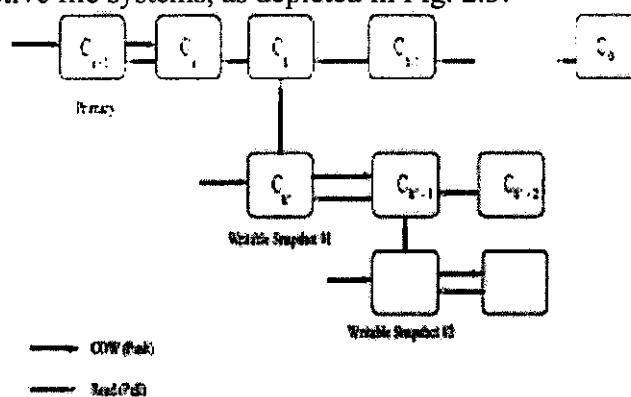
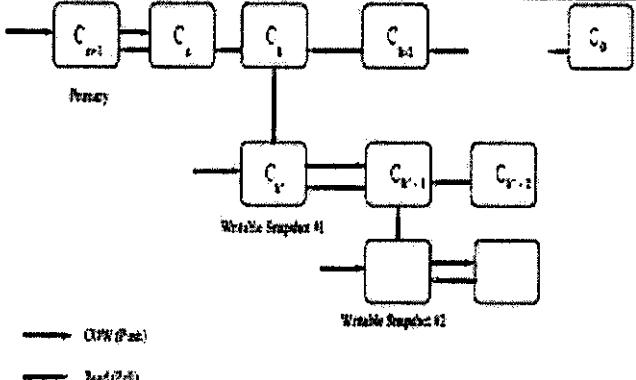
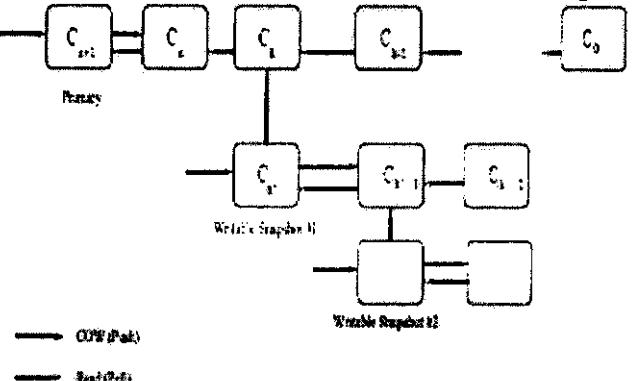
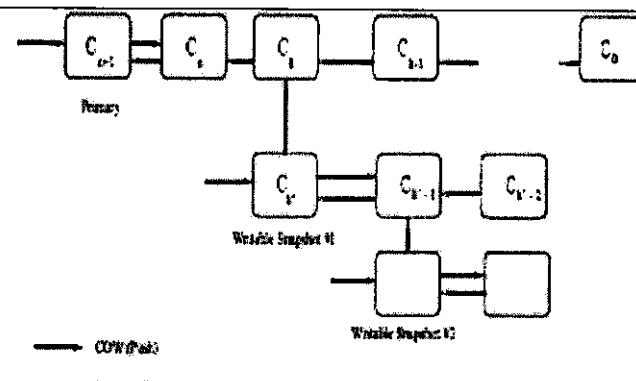


Figure 2.3: Snapshot Tree

Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable

	snapshot C _{k'} initially shares data with snapshot C _k , which represents the state of the original file system at time k. Snapshot C _{k'-1} , representing the state of the new file system at a later time, does not share changed data with snapshot C _{k-1} , which represents the state of the original file system.
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C _{k'} initially shares data with snapshot C _k , which represents the state of the original file system at time k. Snapshot C _{k'-1} , representing the state of the new file system at a later time, does not share changed data with snapshot C _{k-1} , which represents the state of the original file system.
41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches the need to keep snapshots consistent through atomic updates. Fig. 2.3 shows each snapshot to be a past consistent view of active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”

	 <p>Figure 2.3: Snapshot Tree</p>
<p>44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.</p>	<p>Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).</p>
<p>45. A storage system as in claim 43, wherein at least one of the snapshots is converted into a new active file system.</p>	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p>  <p>Figure 2.3: Snapshot Tree</p>
<p>46. A storage system as in claim 45, wherein the one of the snapshots is converted by making the one of the snapshots writable.</p>	<p>Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p>

	 <p>Figure 2.3: Snapshot Tree</p>
<p>47. A storage system as in claim 46, wherein snapshot pointers from any of the active file systems to the new active file system are severed.</p>	<p>Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
<p>48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second</p>	<p>It is inherent that the file system in Siddha Report is executed in conjunction with a computer storage controller. Program and disk implementation of the Siddha Report file system is disclosed in section 4. Thus, the Siddha Report file system is inherently implemented in conjunction with a computer or network interfaces. Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>

active file system, and with changes made to the second active file system not reflected in the first active file system.

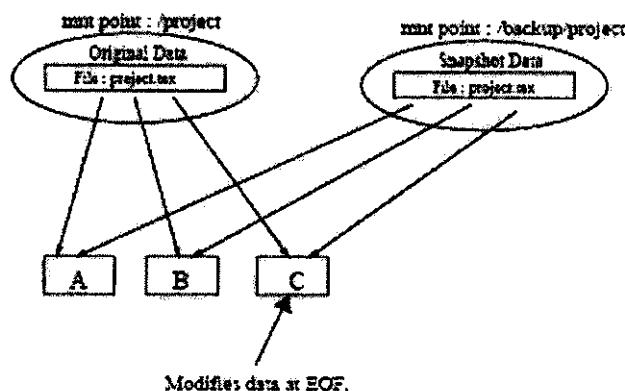


Figure 1.1: Before COW

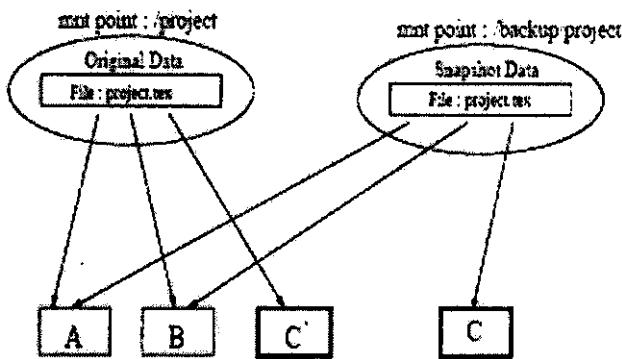


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

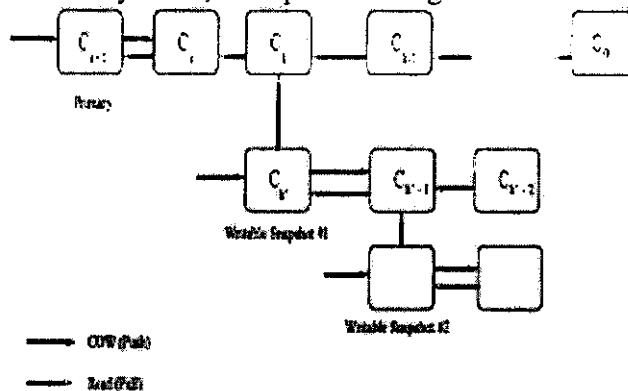


Figure 2.3: Snapshot Tree

Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable

	snapshot C_k' initially shares data with snapshot C_k , which represents the state of the original file system at time k . Snapshot C_{k-1}' , representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1} , which represents the state of the original file system.
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	Siddha Report Fig. 2.3 teaches that there are no pointers between any of the active file systems. Section 2.1.3 further teaches that “[t]he basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”
53. A storage system as in claim 52, wherein each snapshot includes a	Sections 1.1, 2.1.3 and Figs. 1 and 2.3 teach that each snapshot has a hierarchy of metadata that is unique to that snapshot. This is further shown in Fig. 3.4 – each snapshot has its own map. It is

complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	further inherent in the copy-on-write technique that each snapshot and each active file system have unique metadata (such as block maps).
54. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the first active file system, the new snapshot initially sharing data with the first active file system; converting the new snapshot to a third active file system by making the new snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in section 2.1.3: “The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one.”</p>
55. A storage system as in claim 54, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
56. A storage system as in claim 48, wherein the program control further comprises the steps of: making a new snapshot of the second active file system, the new snapshot initially sharing data with the second active file system; converting the new snapshot to a third active file system by making the new	<p>Siddha Report in section 2.1.3 and Fig. 2.3 teaches snapshots of plural active file systems. Section 2.1.3 further teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time.</p> <p>Changes to any of the active file systems are not reflected in the other active file systems, as shown in Fig. 2.3 and taught in</p>

snapshot writable, with changes made to the first active file system or the second active file system not reflected in the third active file system.	section 2.1.3: "The basic idea is to associate an additional snapshot to every snapshot that needs to be mounted writable. All actual writes are then made to this new snapshot and the snapshot chain is modified such that no downstream snapshot references this one."
57. A storage system as in claim 56, wherein when changes are made to the first active file system or the second active file system, modified data is recorded in a location that is not shared with the third active file system.	Due to the copy-on-write nature of snapshots disclosed in Siddha Report, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Fig. 1.
58. An apparatus for operating data storage, the apparatus including means for creating plural active file systems and means for maintaining plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:

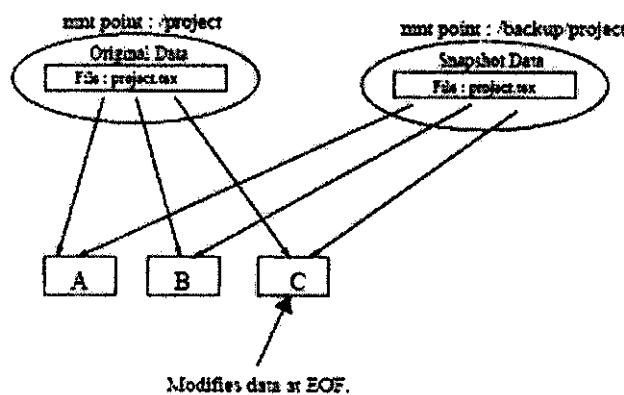


Figure 1.1: Before COW

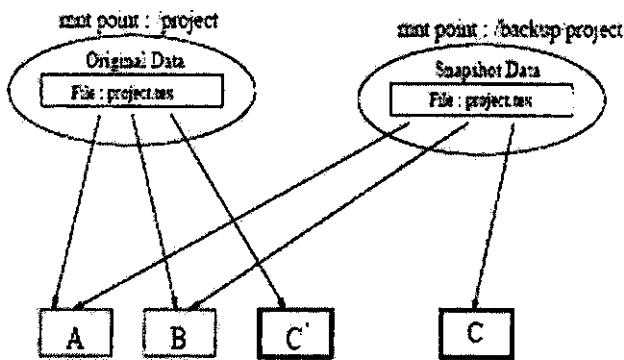


Figure 1.2: After COW

Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:

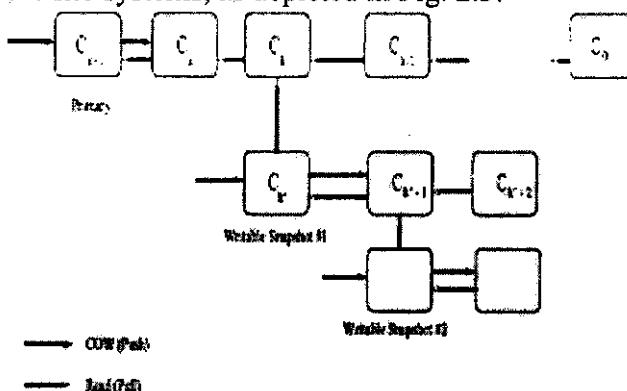
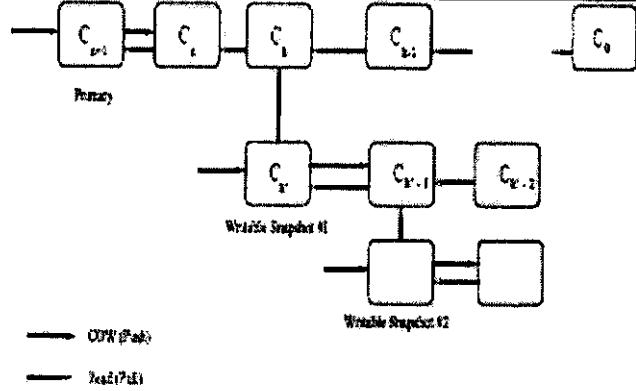


Figure 2.3: Snapshot Tree

Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable

	<p>snapshot C_k' initially shares data with snapshot C_k, which represents the state of the original file system at time k. Snapshot C_{k-1}', representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1}, which represents the state of the original file system.</p>
59. An apparatus for creating plural active file systems, comprising: means for making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and means for converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	<p>Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p> <p>Figure 1.1: Before COW</p> <p>Figure 1.2: After COW</p> <p>Siddha Report section 2.1.3 further teaches that snapshots can be made writable, establishing a diverging snapshot tree of multiple active file systems, as depicted in Fig. 2.3:</p>

	 <p>Figure 2.4: Snapshot Tree</p> <p>Siddha Report teaches that, because of the copy-on-write nature of the snapshots, active file systems formed by mounting snapshots as writable initially share data sets with parent file systems and diverge over time. Thus, in Fig. 2.3, writable snapshot C_k' initially shares data with snapshot C_k, which represents the state of the original file system at time k. Snapshot $C_{k'-1}$, representing the state of the new file system at a later time, does not share changed data with snapshot C_{k-1}, which represents the state of the original file system.</p>
<p>60. A method of operating data storage, comprising: making a snapshot of organizational data of a first active file system, the snapshot pointing to original non-organizational data of the first active file system; storing the snapshot; modifying a first portion of the original non-organizational data of the first active file system in response to a first active file system access request, resulting in a modified first portion being part of first modified non-organizational data of the first active file system; and storing the modified first portion so as not to overwrite the first portion; wherein, after the step of storing the modified</p>	<p>Siddha Report generally and section 2.1.3 specifically teaches a file system with a multiple snapshot capability, wherein the snapshots are established using copy-on-write techniques so as not to overwrite data, as shown in Fig. 1:</p>

first portion, the snapshot points to the original non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data of the first filing system, and the original non-organizational data and the first modified non-organizational data partially overlap.

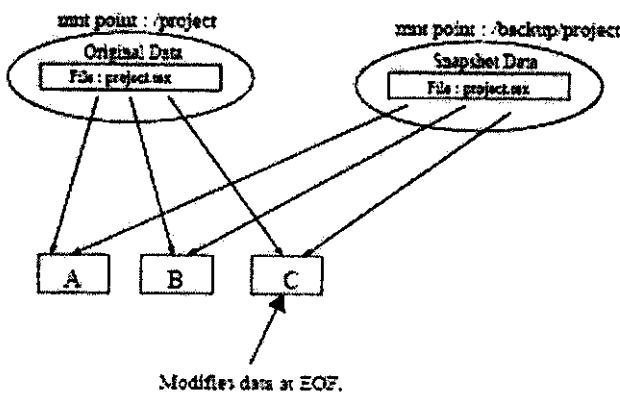


Figure 1.1. Before COW

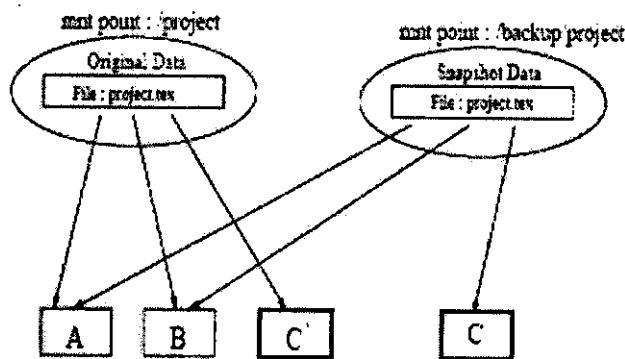
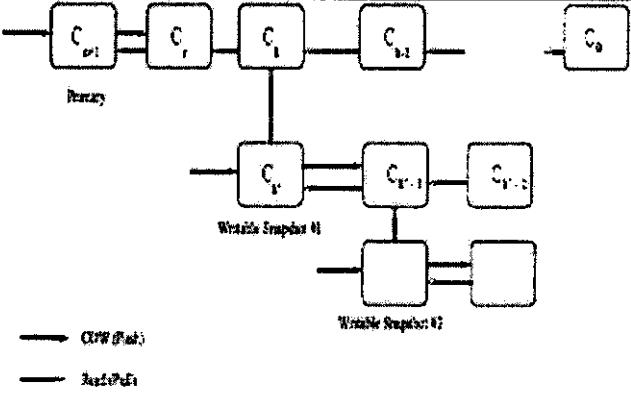


Figure 1.2. After COW

As Fig. 1 shows that the snapshot originally points to data blocks A-C, representative of non-organizational data. When data block C is modified in response to an active file system access request, the snapshot points to the original block, while the active file system organizational data points to the modified data block C'. The active file system and snapshot data sets partially overlap.

61. A method according to claim 60, wherein: the step of storing the snapshot comprises storing the snapshot as a second active filing system; the method further comprising: modifying a second portion of the original non-organizational data in response to a second active

Siddha Report section 2.1.3 teaches that “[w]e can have again read-only snapshots from read-write snapshots and from these read-only snapshots we can again have read-write snapshots. Thus we have a tree like structure as shown in Fig. 2.3.”

<p>file system access request, resulting in a modified second portion being part of second modified non-organizational data of the second active file system; and storing the modified second portion so as not to overwrite the second portion; wherein, after the step of storing the modified second portion, the snapshot points to the second modified non-organizational data, the organizational data of the first active file system point to the first modified non-organizational data, and the first modified non-organizational data and the second modified non-organizational data partially overlap.</p>	 <p>Figure 2.3: Snapshot Tree</p> <p>In Fig. 2.3, snapshot C_k' has been mounted as writable and originally shares data with C_k. As the two active file systems diverge over time, their data sets will partially overlap in the manner shown in Fig. 1.</p>
<p>62. A method according to claim 61, wherein the step of making a snapshot is performed at a consistency point of the first active file system.</p>	<p>Siddha Report sections 1.1 and 2.1.1 teach that snapshots are taken at a consistency point. Fig 2.3 further shows that writable snapshot C_k' is mounted at a consistency point C_k of the original active file system.</p>
<p>63. A method according to claim 62, wherein data is stored in the first and second active file systems using blocks.</p>	<p>Siddha Report section 1.1 and Fig. 1 teach that data is stored using blocks.</p>

Sixth Basis of Invalidity

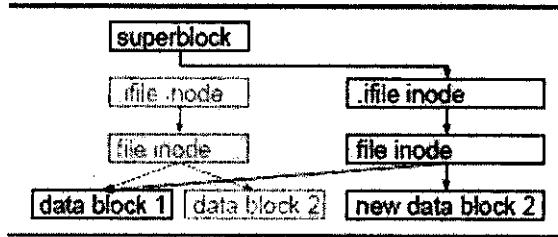
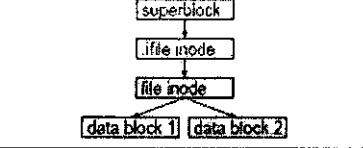
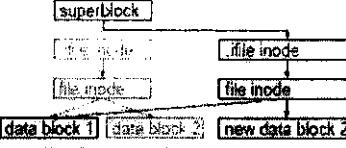
The reference applicable to the sixth basis of invalidity is:

1. C. Czezatke, M. Anton Ertl, *LinLogFS – A log-structured Filesystem For Linux*, Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference, June 18-23, 2000 (hereinafter: “Czezatke”).

The pertinence and manner of applying Czezatke to claims 1-6, 10-12, 14-15, 20-25, 29-33, 39-44, and 48-52 for which re-examination is requested is as follows:

<p>1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Czezatke teaches a Log-Structured File System, which is a method for operating data storage. Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots:</p> <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p> <div data-bbox="612 792 995 960"> <pre> graph TD superblock[superblock] --> fileinode1[file inode] fileinode1 --> fileinode2[file inode] fileinode2 --> datablock1[data block 1] fileinode2 --> datablock2[data block 2] </pre> </div> <div data-bbox="612 971 995 1013" data-label="Caption"> <p>Figure 1. On-disk structure for locating data blocks of a file</p> </div> <div data-bbox="1019 792 1411 960"> <pre> graph TD superblock[superblock] --> fileinode1[file inode] fileinode1 --> fileinode2[file inode] fileinode2 --> datablock1[data block 1] fileinode2 --> newdatablock2[new data block 2] </pre> </div> <div data-bbox="1019 971 1419 1013" data-label="Caption"> <p>Figure 2. Change of the on-disk structure when overwriting data block 2 of the file</p> </div> <p>When the Czezatke file system makes a snapshot or clone, the snapshot or clone inherently initially shares its data with the parent file system. A snapshot directly shares data with its parent active file system. And a clone shares data with its parent snapshot. Thus, when a clone (which is an active file system) is initially created, it will share data with the grandparent active file system to the same extent that its parent snapshot shares data with the grandparent active file system. When the data in an active file system changes, those changes are not reflected in the clone, which is based on a fixed-image snapshot. And, when the data in a clone changes, those changes are not reflected in the parent snapshot or in the active file system from which that snapshot was generated. See Sec. 3.3 (discussing tracking which data blocks are shared with other clones). This feature of writable snapshots makes it possible to explore the “what-if scenarios,” as described by Czezatke in the paragraph reproduced above. The '001 patent specification describes an advantage to writable snapshots that is substantively identical to the description in Czezatke: “It would become possible to make changes to an ‘experimental’ version of the file system.” Col. 1:48-50.</p>

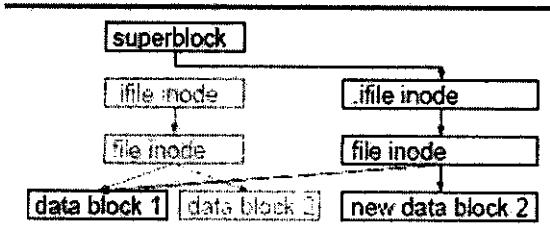
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	The Czezatke file system can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”). Since a writable clone disclosed in Czezatke inherently shares all its data with its parent snapshot at the time of creation, it will also share data with the grandparent active file system to the same extent that the parent snapshot shares data with the grandparent active file system. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones).
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”). Each clone is inherently an image of its respective active file system at a past consistency point.
6. A method as in claim 5, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active	Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:

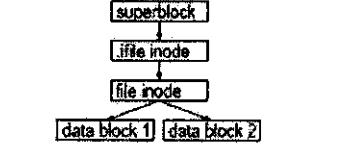
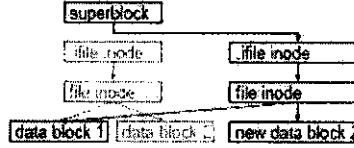
<p>file systems.</p>	 <p>As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.</p>
<p>10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>Czezatke teaches a Log-Structured File System, which is a method for operating data storage. Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots:</p> <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The writable snapshots taught in Czezatke are inherently snapshots that can be made writable.</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;">  <p>Figure 1: On-disk structure for locating data blocks of a file</p> </div> <div style="text-align: center;">  <p>Figure 2: Change of the on-disk structure when overwriting data block 2 of the file</p> </div> </div> <p>When the Czezatke file system makes a snapshot or clone, the snapshot or clone inherently initially shares its data with the parent file system. A snapshot directly shares data with its parent active file system. And a clone shares data with its parent snapshot. Thus, when a clone (which is an active file system) is initially created, it will share data with the grandparent active file system to the same extent that its parent snapshot shares data with the grandparent active file system. When the data in an active file</p>

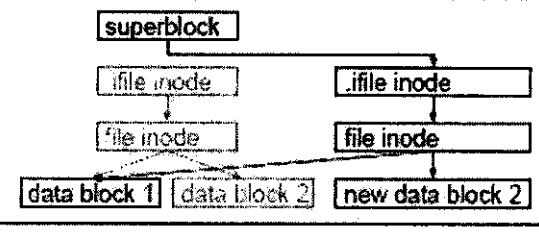
	<p>system changes, those changes are not reflected in the clone, which is based on a fixed-image snapshot. And, when the data in a clone changes, those changes are not reflected in the parent snapshot or in the active file system from which that snapshot was generated. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones). This feature of writable snapshots makes it possible to explore the “what-if scenarios,” as described by Czezatke in the paragraph reproduced above. The '001 patent specification describes an advantage to writable snapshots that is substantively identical to the description in Czezatke: “It would become possible to make changes to an ‘experimental’ version of the file system.” Col. 1:48-50.</p>
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.</p>
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.</p>
14. A method as in claim 10, further comprising the steps of making snapshots of ones of the plural active file systems.	<p>The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”).</p>
15. A method as in claim 14, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system	<p>Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:</p>

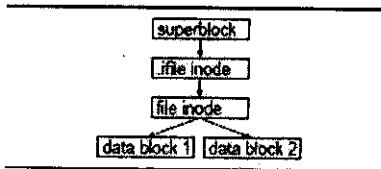
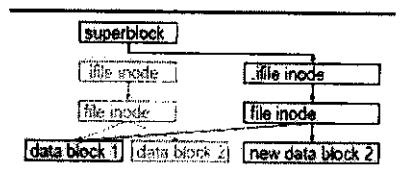
<p>data for the plural active file systems.</p>	<pre> graph TD superblock[superblock] --> .fileinode1[.file inode] superblock --> .fileinode2[.file inode] .fileinode1 --> fileinode1[file inode] .fileinode2 --> fileinode2[file inode] fileinode1 --> dataBlock1[data block 1] fileinode1 --> dataBlock2[data block 2] fileinode2 --> newdataBlock2[new data block 2] </pre> <p>As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.</p>
<p>20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Czezatke teaches a Log-Structured File System software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots:</p> <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <pre> graph TD superblock[superblock] --> .fileinode1[.file inode] superblock --> .fileinode2[.file inode] .fileinode1 --> fileinode1[file inode] .fileinode2 --> fileinode2[file inode] fileinode1 --> dataBlock1[data block 1] fileinode1 --> dataBlock2[data block 2] </pre> <p>Figure 1: On-disk structure for locating data blocks of a file</p> </div> <div style="text-align: center;"> <pre> graph TD superblock[superblock] --> .fileinode1[.file inode] superblock --> .fileinode2[.file inode] .fileinode1 --> fileinode1[file inode] .fileinode2 --> fileinode2[file inode] fileinode1 --> dataBlock1[data block 1] fileinode1 --> newdataBlock2[new data block 2] </pre> <p>Figure 2: Change of the on-disk structure when overwriting data block 2 of the file</p> </div> </div> <p>When the Czezatke file system makes a snapshot or clone, the snapshot or clone inherently initially shares its data with the parent file system. A snapshot directly shares data with its parent active file system. And a clone shares data with its parent snapshot. Thus, when a clone (which is an active file system) is initially created, it will share data with the grandparent active file system to the same extent that its parent snapshot shares data with the grandparent active file system. When the data in an active file system changes, those changes are not reflected in the clone, which is based on a fixed-image snapshot. And, when the data in a clone changes, those changes are not reflected in the parent snapshot or</p>

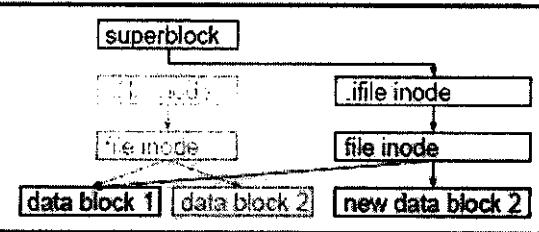
	<p>in the active file system from which that snapshot was generated. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones). This feature of writable snapshots makes it possible to explore the “what-if scenarios,” as described by Czezatke in the paragraph reproduced above. The ‘001 patent specification describes an advantage to writable snapshots that is substantively identical to the description in Czezatke: “It would become possible to make changes to an ‘experimental’ version of the file system.” Col. 1:48-50.</p>
<p>21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.</p>	<p>The Czezatke file system can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”). Since a writable clone disclosed in Czezatke inherently shares all its data with its parent snapshot at the time of creation, it will also share data with the grandparent active file system to the same extent that the parent snapshot shares data with the grandparent active file system. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones).</p>
<p>22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.</p>
<p>23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.</p>
<p>24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each</p>	<p>The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”). Each clone is inherently an image of its respective active file system at a</p>

snapshot forming an image of its respective active file system at a past consistency point.	past consistency point.
25. A method as in claim 24, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	<p>Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:</p>  <p>As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.</p>
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the	<p>Czezatke teaches a Log-Structured File System software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots:</p> <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The writable snapshots taught in Czezatke are inherently snapshots that can be made writable.</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p>

first active file system.	 <p>Figure 1: On-disk structure for locating data blocks of a file</p>	 <p>Figure 2: Change of the on-disk structure when overwriting data block 2 of the file</p>
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.	
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.	

not shared with the first active file system.	
32. A memory as in claim 29, wherein the instructions further comprise the step of severing any snapshot pointers from the first active file system to the second active file system.	The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”).
33. A memory as in claim 29, wherein the instructions further comprise the steps of making snapshots of ones of the plural active file systems.	Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:  <pre> graph TD superblock[superblock] --> ifile1[ifile inode] superblock --> ifile2[ifile inode] ifile1 --> file1[file inode] ifile2 --> file2[file inode] file1 --> data1[data block 1] file2 --> data2[data block 2] file2 -.-> newdata[new data block 2] </pre>
	As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access	Czezatke teaches a Log-Structured File System implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. <i>See Sec. 7</i> (describing computer and network setup used to implement the file system). Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots: <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p>

<p>data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	 <p>Figure 1: On-disk structure for locating data blocks of a file</p>  <p>Figure 2: Change of the on-disk structure when overwriting data block 2 of the file.</p> <p>When the Czezatke file system makes a snapshot or clone, the snapshot or clone inherently initially shares its data with the parent file system. A snapshot directly shares data with its parent active file system. And a clone shares data with its parent snapshot. Thus, when a clone (which is an active file system) is initially created, it will share data with the grandparent active file system to the same extent that its parent snapshot shares data with the grandparent active file system. When the data in an active file system changes, those changes are not reflected in the clone, which is based on a fixed-image snapshot. And, when the data in a clone changes, those changes are not reflected in the parent snapshot or in the active file system from which that snapshot was generated. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones). This feature of writable snapshots makes it possible to explore the “what-if scenarios,” as described by Czezatke in the paragraph reproduced above. The '001 patent specification describes an advantage to writable snapshots that is substantively identical to the description in Czezatke: “It would become possible to make changes to an ‘experimental’ version of the file system.” Col. 1:48-50.</p>
<p>40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.</p>	<p>The Czezatke file system can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See Sec. 4</i> (discussing implementing “writable, clonable clones”). Since a writable clone disclosed in Czezatke inherently shares all its data with its parent snapshot at the time of creation, it will also share data with the grandparent active file system to the same extent that the parent snapshot shares data with the grandparent active file system. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones).</p>
<p>41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.</p>

in the first active file system in a location that is not shared with the second active file system.	
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4 (discussing implementing “writable, clonable clones”).</i> Each clone is inherently an image of its respective active file system at a past consistency point.
44. A storage system as in claim 43, wherein each snapshot includes a complete hierarchy for file system data, separate and apart from active file system data for the plural active file systems.	Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:
	 <p>As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.</p>
48. A storage system, comprising: at least one	Czezatke teaches a Log-Structured File System implemented on a computer having at least one disk storage device and connected to

<p>storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>a network for receiving and sending information. <i>See Sec. 7</i> (describing computer and network setup used to implement the file system). Czezatke further teaches maintaining plural active file systems through the mechanism of cloning or writable snapshots:</p> <ul style="list-style-type: none"> • Cheap cloning (writable snapshots) of the whole file system can be used for exploring what-if scenarios while writing to the other clone of the file system (in addition to consistent backups). <p>(see 1. Introduction – page 1)</p> <p>The writable snapshots taught in Czezatke are inherently snapshots that can be made writable.</p> <p>The Czezatke file system is implemented using copy-on-write techniques, where new data never overwrites old data:</p> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <pre> graph TD superblock[superblock] --> fileinode1[file inode] fileinode1 --> fileinode2[file inode] fileinode2 --> dataBlock1[data block 1] fileinode2 --> dataBlock2[data block 2] </pre> </div> <div style="text-align: center;"> <pre> graph TD superblock[superblock] --> fileinode1[file inode] fileinode1 --> fileinode2[file inode] fileinode2 --> dataBlock1[data block 1] fileinode2 --> newDataBlock2[new data block 2] </pre> </div> </div> <p>Figure 1: On-disk structure for locating data blocks of a file</p> <p>Figure 2: Change of the on-disk structure when overwriting data block 2 of the file</p> <p>When the Czezatke file system makes a snapshot or clone, the snapshot or clone inherently initially shares its data with the parent file system. A snapshot directly shares data with its parent active file system. And a clone shares data with its parent snapshot. Thus, when a clone (which is an active file system) is initially created, it will share data with the grandparent active file system to the same extent that its parent snapshot shares data with the grandparent active file system. When the data in an active file system changes, those changes are not reflected in the clone, which is based on a fixed-image snapshot. And, when the data in a clone changes, those changes are not reflected in the parent snapshot or in the active file system from which that snapshot was generated. <i>See Sec. 3.3</i> (discussing tracking which data blocks are shared with other clones). This feature of writable snapshots makes it possible to explore the “what-if scenarios,” as described by Czezatke in the paragraph reproduced above. The ’001 patent specification describes an advantage to writable snapshots that is substantively identical to the description in Czezatke: “It would become possible to make changes to an ‘experimental’ version of the file system.” Col. 1:48-50.</p>
<p>49. A storage system as in</p>	<p>Due to the copy-on-write nature of snapshots and active file</p>

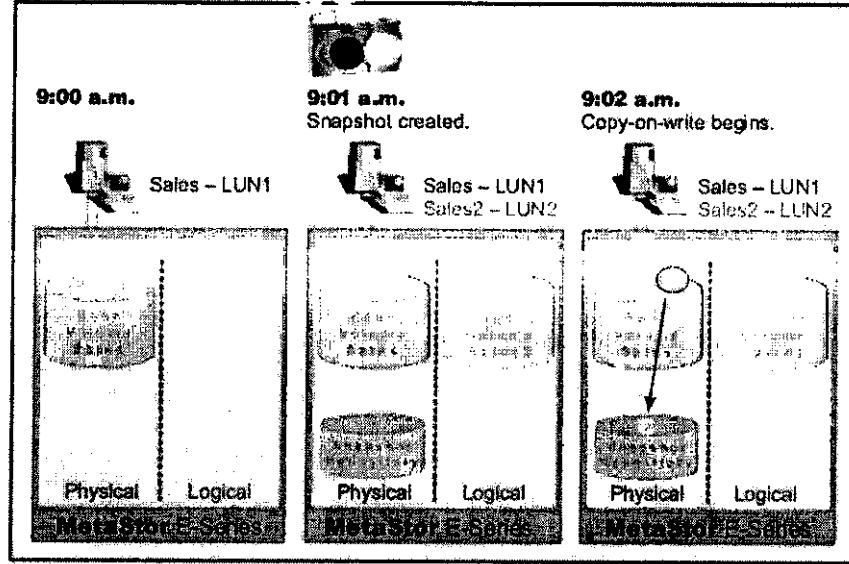
claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Czezatke, modified data in different active file systems is inherently written in new locations so as to not overwrite other data, as shown in Figs. 1 and 2.
51. A storage system as in claim 48, wherein the program control further comprises the step of severing any snapshot pointers from the first active file system to the second active file system.	The Czezatke file system discloses making snapshots (clones) of writable snapshots, each of which is an active file system. <i>See Sec. 4 (discussing implementing “writable, clonable clones”).</i>
52. A storage system as in claim 48, wherein the program control further comprises the steps of making snapshots of ones of the plural active file systems.	<p>Due to the copy-on-write nature of the Czezatke file system, each snapshot includes a complete hierarchy of file system data, such as the ifile inode and lower level inodes, that is separate and apart from active file system data, as illustrated in Fig. 2:</p> <div data-bbox="592 1368 1142 1615"> <pre> graph TD superblock[superblock] --> ifile1[ifile inode] superblock --> ifile2[ifile inode] ifile1 --> file1[file inode] ifile2 --> file2[file inode] file1 --> data1[data block 1] file1 --> data2[data block 2] file2 --> newdata[new data block 2] </pre> </div> <p>As shown above, if a snapshot is taken of the previous consistent state of the file system shown on the left, it would maintain a unique set of file system data. The same inherently holds true for the relationship between each snapshot and each writable clone in Czezatke – each has a unique set of file system data, separate and apart from other active file systems.</p>

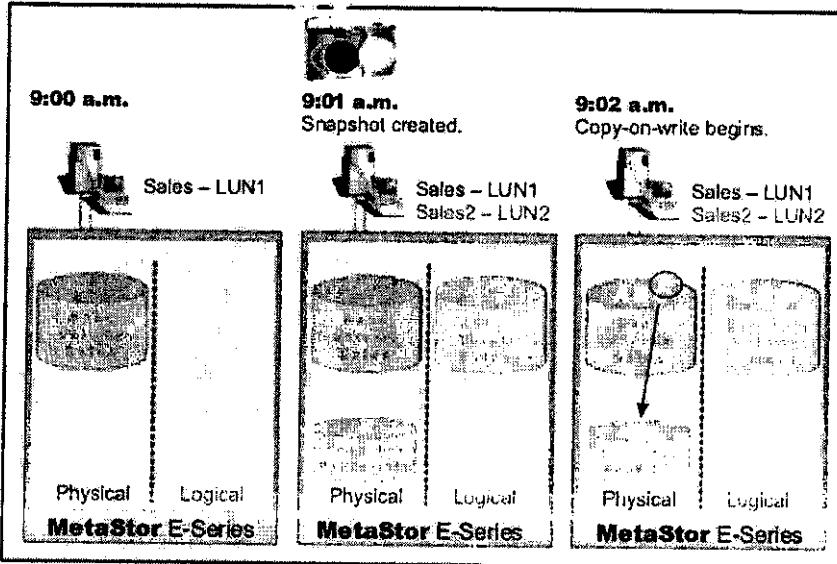
Seventh Basis of Invalidity

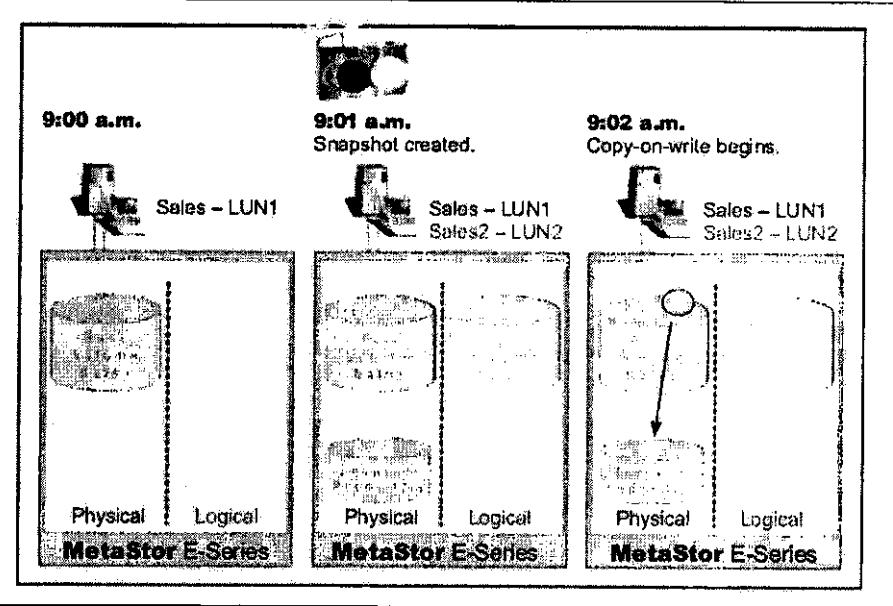
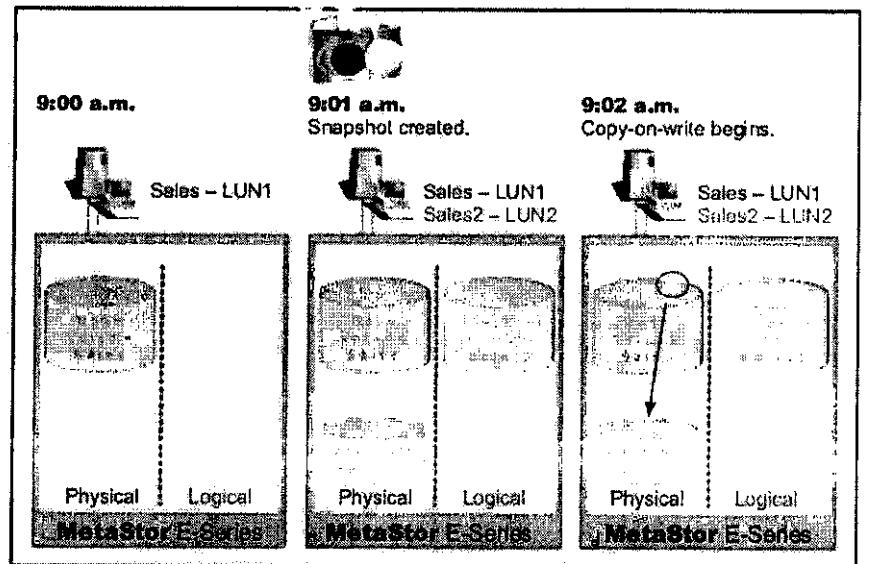
The reference applicable to the seventh basis of invalidity is:

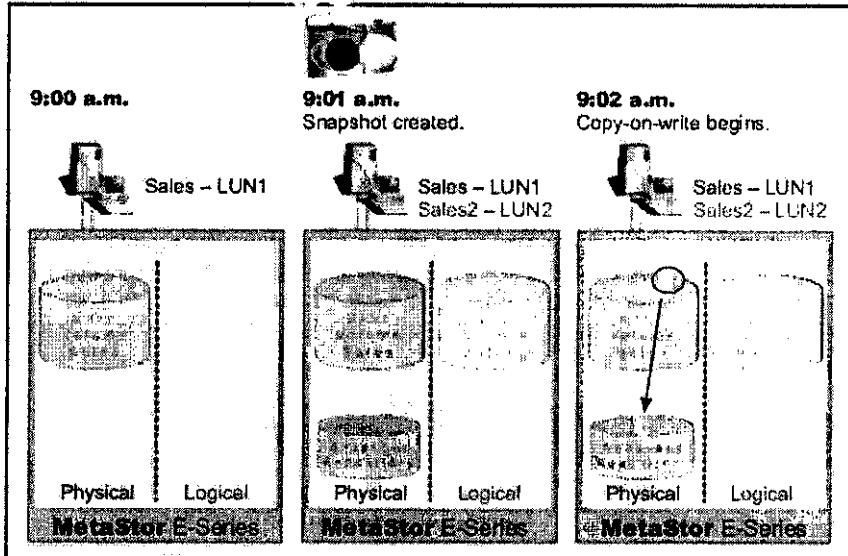
1. The Enterprise Challenge Served By Snapshot, LSI Logic Whitepaper, 2001 (hereinafter: "LSI Logic Whitepaper").

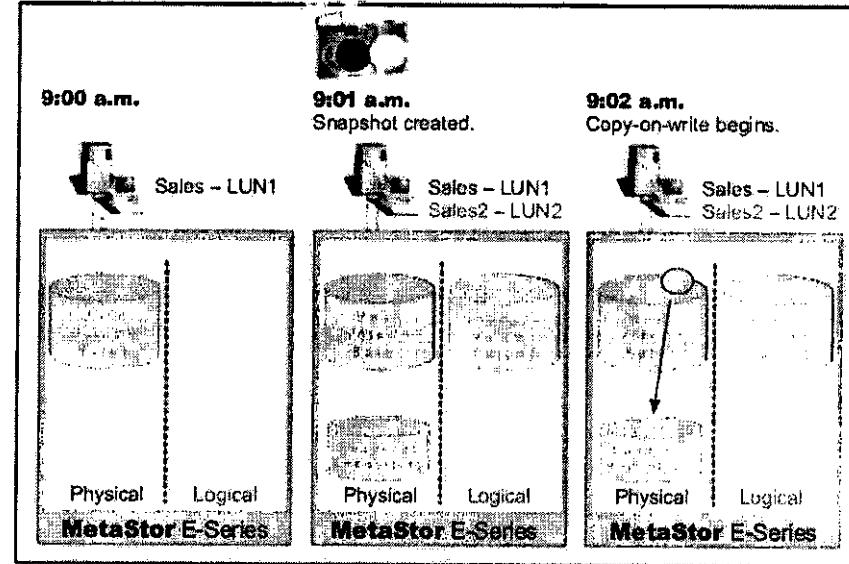
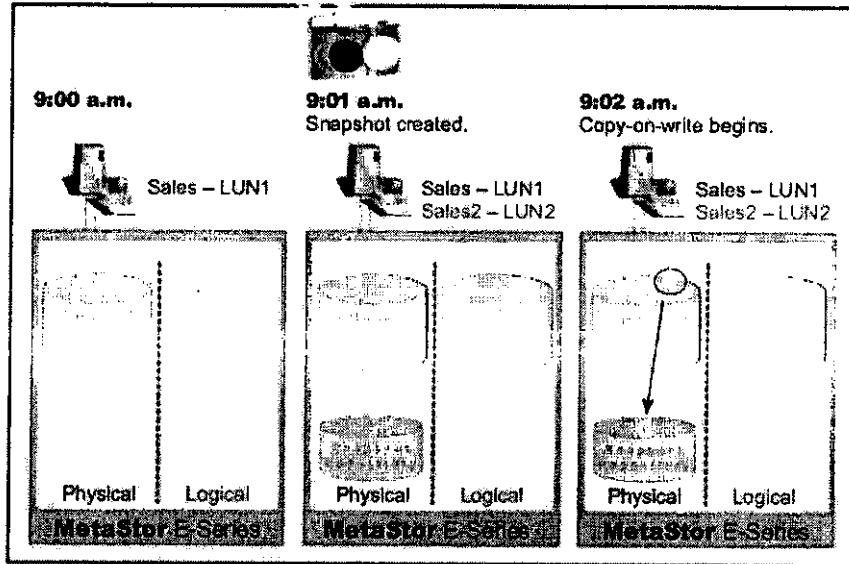
The pertinence and manner of applying LSI Logic Whitepaper to claims 1-5, 10-12, 20-24, 29-31, 39-43 and 48-50 for which re-examination is requested is as follows:

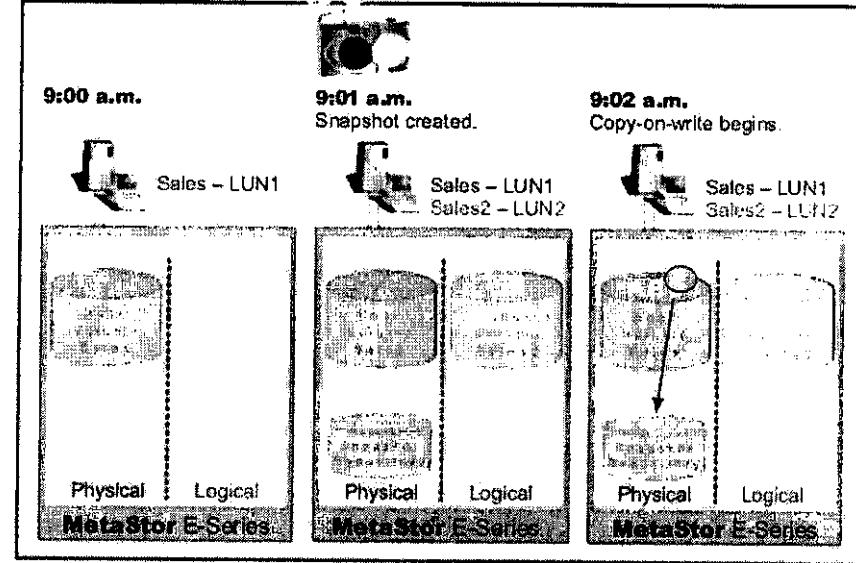
<p>1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>LSI Logic Whitepaper discloses a method of making writable snapshots of logical volumes, wherein such a volume is representative of a file system: The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change. however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature. (see page 4)</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume."</p>  <p>Thus, when a writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system). As data is written to the writable snapshot, it will diverge from the base volume, with changes in one not reflected in the other.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

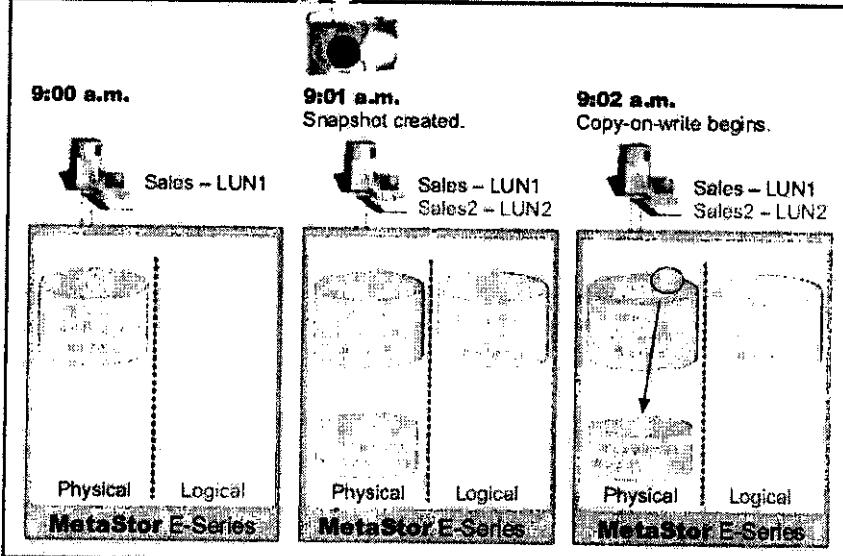
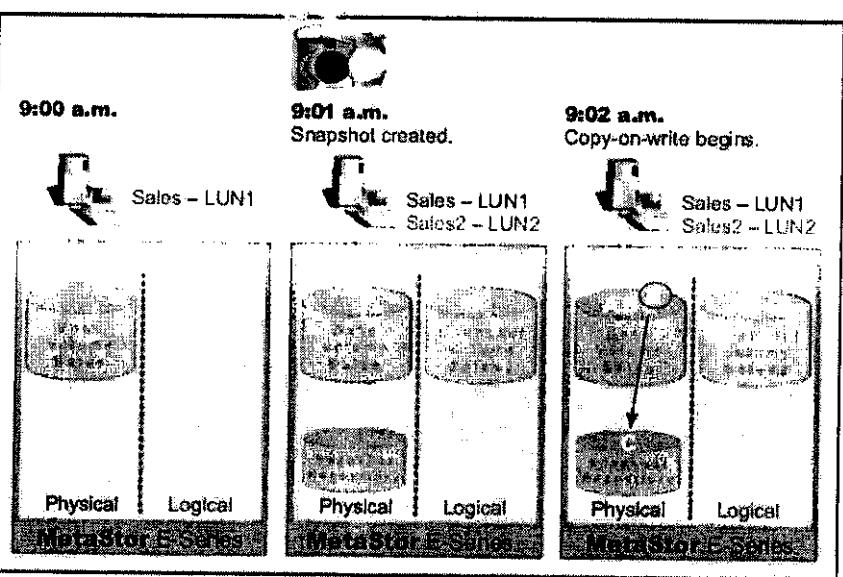
	the other.
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	The LSI Logic snapshot feature can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See p. 4.</i> When the writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system).
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:
	

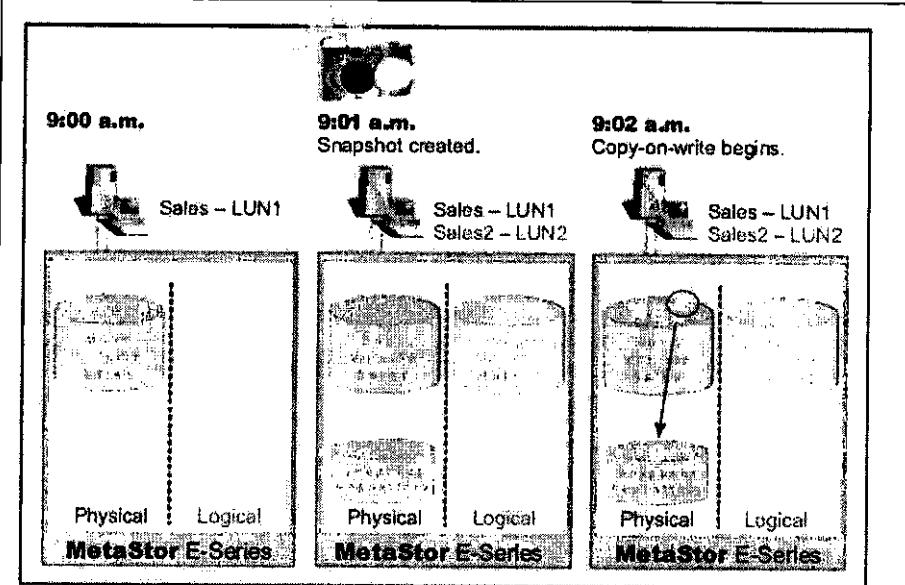
	
<p>5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.</p>	<p>The LSI Logic Whitepaper file discloses making snapshots (clones) of active file systems, referred to as base volumes:</p>  <p>Each snapshot is an image of its respective active file system at a past consistency point. For example, in the diagram above, the snapshot is an image of the base volume in a consistent state at 9:01 a.m.</p>
<p>10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file</p>	<p>LSI Logic Whitepaper discloses a method of making writable snapshots of logical volumes, wherein such a volume is representative of a file system:</p>

<p>system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature. (see page 4)</p> <p>Because the SANtricity snapshots can be written to, they are converted to active file systems by making them writable.</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume:"</p> 
<p>11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p>

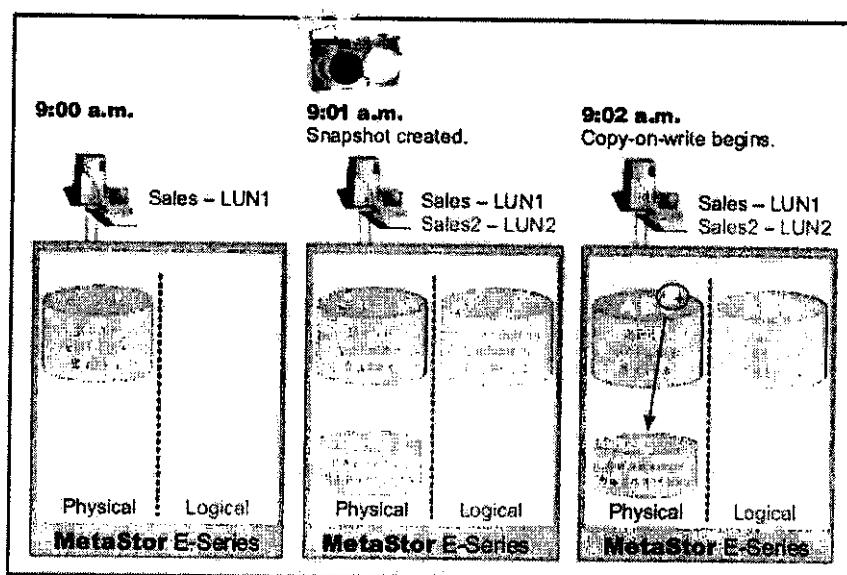
active file system.	
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage,	<p>LSI Logic Whitepaper teaches a snapshot feature implemented in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. LSI Logic Whitepaper discloses making writable snapshots of logical volumes, wherein such a volume is representative of a file system:</p>

<p>the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature. (see page 4)</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume:"</p> 
<p>21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.</p>	<p>The LSI Logic snapshot feature can make writable snapshots (i.e., a "second active file system") from a snapshot of an active file system. See p. 4. When the writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system).</p>
<p>22. A memory as in claim 21, wherein</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different</p>

<p>when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.</p>	<p>active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
<p>23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
<p>24. A memory as in claim 20, wherein snapshots are made of</p>	<p>The LSI Logic Whitepaper file discloses making snapshots (clones) of active file systems, referred to as base volumes:</p>

<p>ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.</p>	
<p>29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in</p>	<p>LSI Logic Whitepaper teaches a snapshot feature implemented in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. LSI Logic Whitepaper discloses making writable snapshots of logical volumes, wherein such a volume is representative of a file system: The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature. (see page 4)</p> <p>Because the SANtricity snapshots can be written to, they are converted to active file systems by making them writable.</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume."</p>

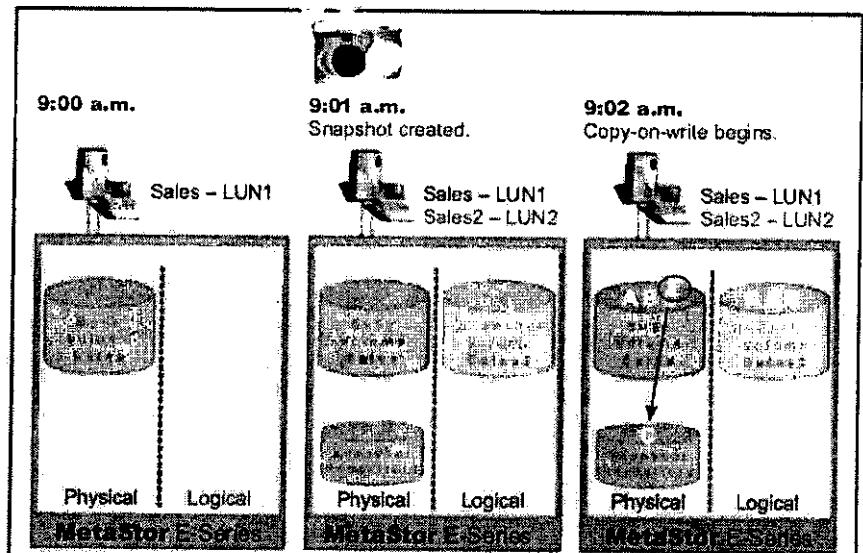
the first active file system.

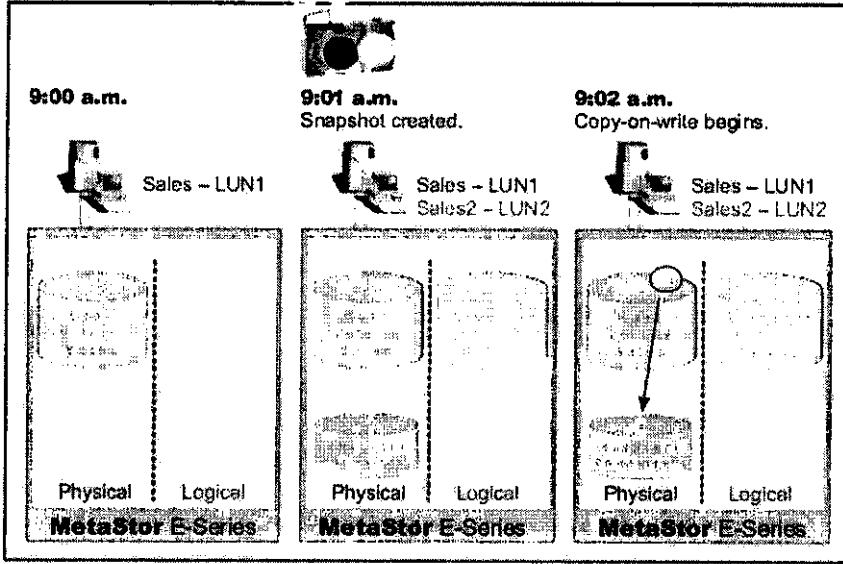


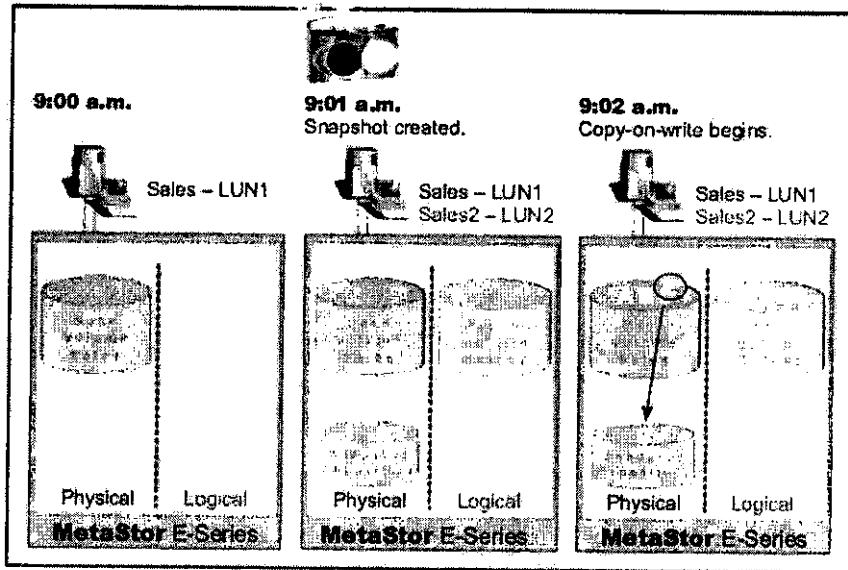
Thus, when a writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system). As data is written to the writable snapshot, it will diverge from the base volume, with changes in one not reflected in the other.

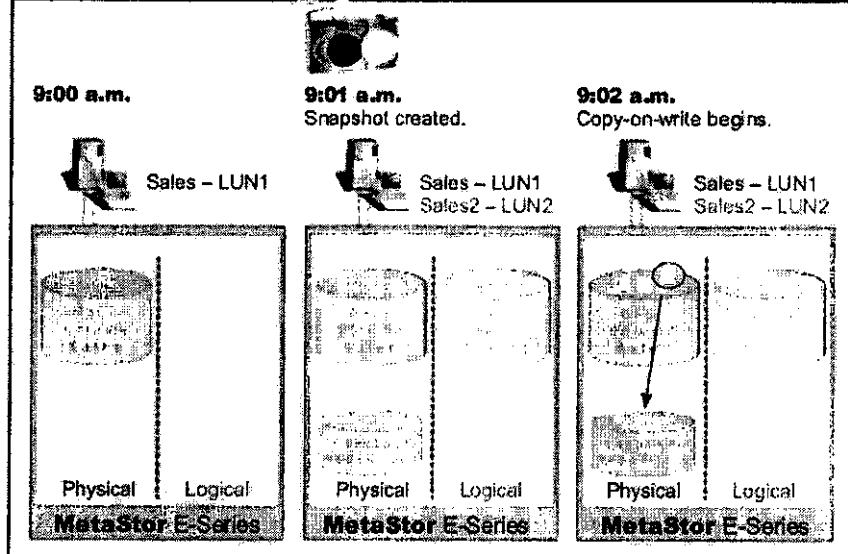
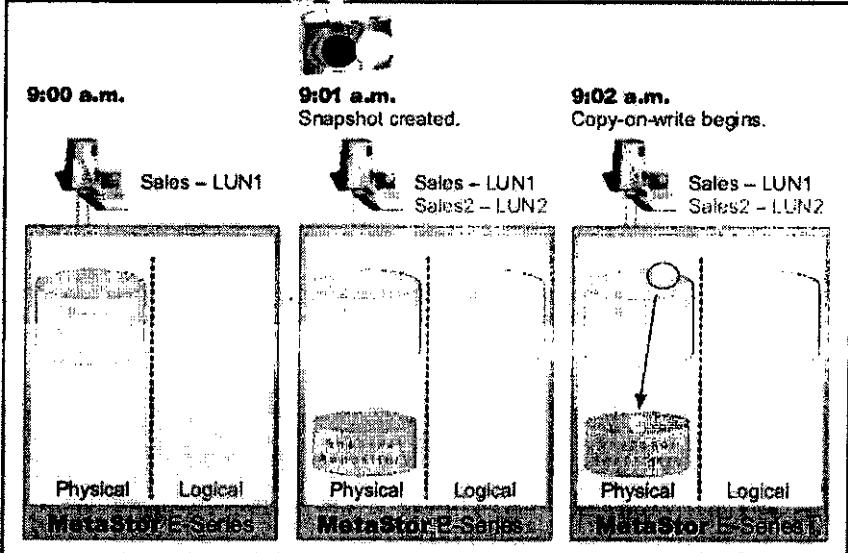
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.

Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:



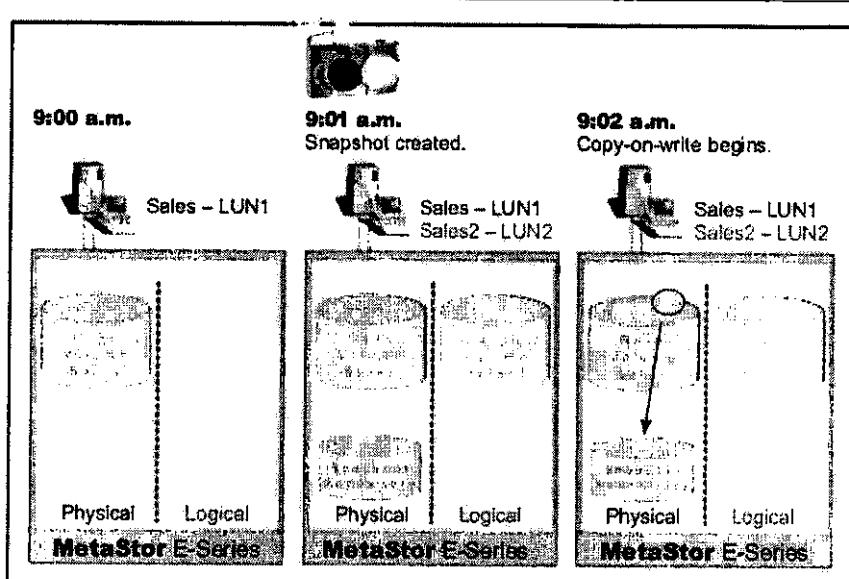
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the	<p>LSI Logic Whitepaper teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. LSI Logic Whitepaper discloses making writable snapshots of logical volumes, wherein such a volume is representative of a file system:</p> <p>The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature.</p> <p>(see page 4)</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume."</p>

<p>active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	
<p>40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.</p>	<p>The LSI Logic snapshot feature can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See p. 4.</i> When the writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system).</p>
<p>41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p>

	
<p>42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.</p>	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
<p>43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming</p>	<p>The LSI Logic Whitepaper file discloses making snapshots (clones) of active file systems, referred to as base volumes;</p>

<p>an image of its respective active file system at a past consistency point.</p>	<p>The diagram illustrates the creation of a snapshot and the start of copy-on-write for a file system. It shows three time points:</p> <ul style="list-style-type: none"> 9:00 a.m.: A storage system (MetaStor E-Series) contains a single LUN labeled "Sales - LUN1". This LUN is divided into a "Physical" section and a "Logical" section. 9:01 a.m. Snapshot created.: A new LUN labeled "Sales2 - LUN2" is created, which is a copy of the "Sales - LUN1" at the 9:00 a.m. point. It also has "Physical" and "Logical" sections. The original "Sales - LUN1" is now labeled "Sales - LUN1" and "Sales2 - LUN2". 9:02 a.m. Copy-on-write begins.: A file is being written to the "Logical" section of "Sales2 - LUN2". A dashed arrow points from this write operation to the "Physical" section of "Sales - LUN1", indicating that the write is being copied to the base volume.
<p>48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file</p>	<p>LSI Logic Whitepaper teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. LSI Logic Whitepaper discloses making writable snapshots of logical volumes, wherein such a volume is representative of a file system:</p> <p>The SANtricity snapshot feature is able to read, write and copy the snapshot. The writes, or updates to the snapshot, are handled in the repository. If a write occurs to the snapshot, it overwrites the PIT image with the change, however, this is a valuable capability that opens up new techniques for creating immediate test and small data mining environments. This is also a distinguishing capability of LSI Logic's snapshot feature.</p> <p>(see page 4)</p> <p>Because the SANtricity snapshots can be written to, they are converted to active file systems by making them writable.</p> <p>When the LSI Logic SANtricity snapshot feature makes a writable snapshot, the snapshot inherently initially shares its data with the parent file system. The writable snapshot directly shares and data with its parent active file system, referred to as the "base volume."</p>

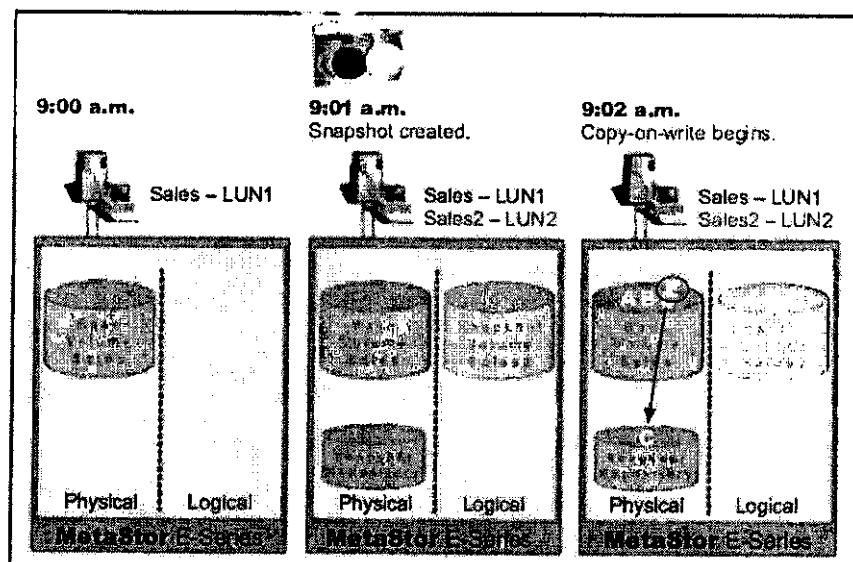
system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.

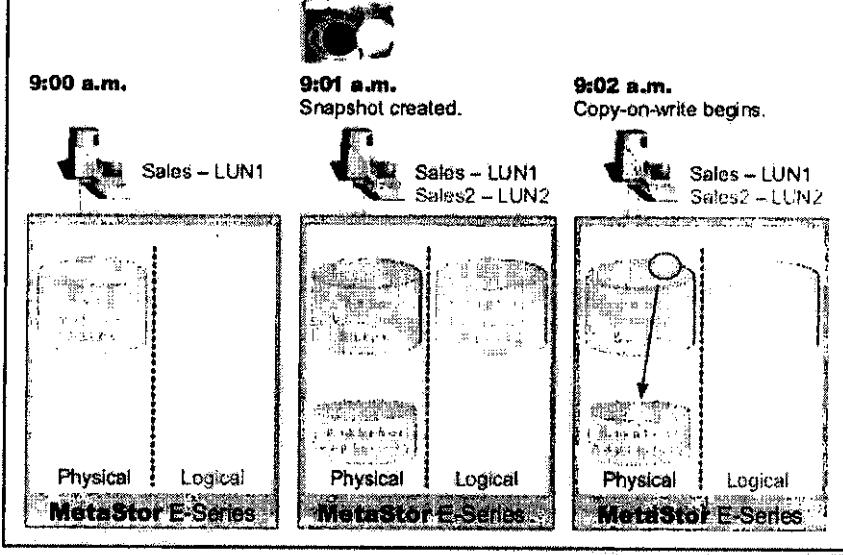


Thus, when a writable snapshot (which is an active file system) is initially created, it will share data with its base volume (which remains an active file system). As data is written to the writable snapshot, it will diverge from the base volume, with changes in one not reflected in the other.

49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.

Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:



50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	<p>Due to the copy-on-write nature of snapshots and active file systems disclosed in the LSI Logic Whitepaper, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data, as shown below:</p> 
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Eighth Basis of Invalidity

The reference applicable to the eighth basis of invalidity is:

1. N. Osorio and B. Lee, *Guidelines for Using Snapshot Storage Systems for Oracle Databases*, version 1 dated August 28, 2000 (hereinafter: "Osorio").

The pertinence and manner of applying Osorio to claims 1-5, 10-12, 20-24, 29-31, 39-43 and 48-50 for which re-examination is requested is as follows.

<p>1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a "copy image of storage devices or file systems." P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other active file system: "When a second host can read</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	and write the snapshot image, the second host can start a second Oracle instance. . . . For this usage, the snapshot area essentially is a new database started with a copy of the original database.”
2. A method as in claim 1, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	Osorio discloses that its snapshot feature can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See</i> p. 7. When the writable snapshot (which is an active file system) is initially created, it will share data with its parent file system (which remains an active file system).
3. A method as in claim 2, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
4. A method as in claim 2, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
5. A method as in claim 1, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Each snapshot in Osorio is inherently an image of its respective active file system at a past consistency point.
10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active	Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a “copy image of storage devices or file systems.” P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other

file system not reflected in the first active file system.	active file system: "When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . For this usage, the snapshot area essentially is a new database started with a copy of the original database." The snapshot is converted to a second active file system by making it writable.
11. A method as in claim 10, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
12. A method as in claim 10, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Osorio teaches a snapshot feature implemented in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a "copy image of storage devices or file systems." P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other active file system: "When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . For this usage, the snapshot area essentially is a new database started with a copy of the original database."
21. A memory as in claim 20, wherein when a second active file system is created based on a first active file system, the first	The Osorio snapshot feature can make writable snapshots (i.e., a "second active file system") from a snapshot of an active file system. See p. 7.

active file system and the second active file system initially share data.	When the writable snapshot (which is an active file system) is initially created, it will share data with its parent file system (which remains an active file system).
22. A memory as in claim 21, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
23. A memory as in claim 21, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
24. A memory as in claim 20, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Each snapshot in Osorio is inherently an image of its respective active file system at a past consistency point.
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Osorio teaches a snapshot feature implemented in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a "copy image of storage devices or file systems." P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other active file system: "When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . For this usage, the snapshot area essentially is a new database started with a copy of the original

	database.” The snapshot is converted to a second active file system by making it writable.
30. A memory as in claim 29, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
31. A memory as in claim 29, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Osorio teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a “copy image of storage devices or file systems.” P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other active file system: “When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . . For this usage, the snapshot area essentially is a new database started with a copy of the original database.”
40. A storage system as in claim 39, wherein when a second active file system is created based on a first active file system, the first active file system and the second active file system initially share data.	The Osorio snapshot feature can make writable snapshots (i.e., a “second active file system”) from a snapshot of an active file system. <i>See</i> p. 7. When the writable snapshot (which is an active file system) is initially created, it will share data with its parent file system (which remains an active file system).

41. A storage system as in claim 40, wherein when changes are made to the first active file system, modified data is recorded in the first active file system in a location that is not shared with the second active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
42. A storage system as in claim 40, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.
43. A storage system as in claim 39, wherein snapshots are made of ones of the plural active file systems, each snapshot forming an image of its respective active file system at a past consistency point.	Each snapshot in Osorio is inherently an image of its respective active file system at a past consistency point.
48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Osorio teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. Osorio discloses a method for operating data storage including maintaining file systems using copy-on-write snapshots that can be written to. Pp. 3-4 disclose copy-on-write snapshots as a "copy image of storage devices or file systems." P. 7 further teaches that the snapshot can be converted into another active file system that initially shares data with the original system, but diverges over time, with changes made to each of the active file systems not reflected in the other active file system: "When a second host can read and write the snapshot image, the second host can start a second Oracle instance. . . For this usage, the snapshot area essentially is a new database started with a copy of the original database." The snapshot is converted to a second active file system by making it writable.
49. A storage system as in claim 48, wherein when changes are made to the first active file system, modified data is	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently

recorded in the first active file system in a location that is not shared with the second active file system.	written in new locations not shared with the other active file systems so as to not overwrite other data.
50. A storage system as in claim 48, wherein when changes are made to the second active file system, modified data is recorded in the second active file system in a location that is not shared with the first active file system.	Due to the copy-on-write nature of snapshots and active file systems disclosed in Osorio, modified data in different active file systems is inherently written in new locations not shared with the other active file systems so as to not overwrite other data.

Ninth Basis of Invalidity

The reference applicable to the ninth basis of invalidity is:

1. U.S. Pat. 6,341,341 to Grummon et al. (hereinafter "Grummon").

The pertinence and manner of applying Grummon to claims 1, 10, 20, 29, 39 and 48 for which re-examination is requested is as follows.

1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Grummon discloses a file system using copy-on-write snapshots and having writable snapshots. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63.
10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Grummon discloses a file system using copy-on-write snapshots and having a writable snapshot. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63. The snapshot is converted to a second active file system by making it writable.
20. A memory storing information	Grummon teaches a snapshot feature implemented

<p>including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Grummon discloses a file system using copy-on-write snapshots and having writable snapshots. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63.</p>
<p>29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>Grummon teaches a snapshot feature implemented in software stored on a memory, wherein the software comprises instructions executable by a processor to operate data storage. Grummon discloses a file system using copy-on-write snapshots and having a writable snapshot. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63. The snapshot is converted to a second active file system by making it writable.</p>
<p>39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Grummon teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. Grummon discloses a file system using copy-on-write snapshots and having writable snapshots. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63.</p>

<p>48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>Grummon teaches a snapshot feature implemented on a computer having at least one disk storage device and connected to a network for receiving and sending information. Grummon discloses a file system using copy-on-write snapshots and having a writable snapshot. Fig. 3 discloses a file system 300. Read-Write On-Line Container 310 represents the active file system while Read-Write Snapshot Container 308 represents the writable snapshot. Col. 7: 11-16. The snapshot and active file system initially share data. <i>See e.g.</i> Col. 6: 37-47. As data is written to container 308, changes are not shared. <i>See e.g.</i> Col. 7:17-63. The snapshot is converted to a second active file system by making it writable.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tenth Basis of Invalidity

The reference applicable to the tenth basis of invalidity is:

1. U.S. Pat. 5,675,802 to Allen et al. (hereinafter “Allen”).

The pertinence and manner of applying Allen to claims 1, 10, 20, 29, 39 and 48 for which re-examination is requested is as follows.

<p>1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.
20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.
29. A memory storing information including instructions, the instructions executable by a processor to create plural active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file	Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree

system not reflected in the first active file system.	are not reflected in the file trees with which the first tree initially shares data.
39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.	Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.
48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.	Allen teaches a file system for viewing selected versions of the files stored in a versioned object base (VOB). Col. 6:16-18, Fig. 2. A virtual file system is provided that allows users to have a private view of the files. Col. 6:49-58. Allen further teaches “branching” of the file trees, with each of the branches undergoing independent evolution from parent and sister branches. Col. 6:59-7:22, Fig. 3. As shown in Fig. 3, a branch initially shares data with the parent, but diverges over time, with changes not shared with the other branches. Therefore, Allen teaches plural active file trees, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.

Eleventh Basis of Invalidity

The reference applicable to the eleventh basis of invalidity is:

1. U.S. Pat. 6,795,966 to Lim et al. (hereinafter: “Lim”).

The pertinence and manner of applying Lim to claims 1, 10, 20, 29, 39 and 48 for which re-examination is requested is as follows.

<p>1. A method of operating data storage, the method including maintenance of plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.</p>
<p>10. A method of creating plural active file systems, comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data. A checkpoint is converted to an active file system by making it writable and using it as an initial state. <i>See e.g.</i> Col. 7: 6-18, 8:8-14.</p>
<p>20. A memory storing information including instructions, the instructions executable by a processor to operate data storage, the instructions comprising steps to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.</p>
<p>29. A memory storing information including instructions, the instructions executable by a processor to create plural</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can</p>

<p>active file systems, the instructions comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes made to the second active file system not reflected in the first active file system.</p>	<p>be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data. A checkpoint is converted to an active file system by making it writable and using it as an initial state. <i>See e.g.</i> Col. 7: 6-18, 8:8-14.</p>
<p>39. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to maintain plural active file systems, wherein each of the active file systems initially access data shared with another of the active file systems, and wherein changes made to each of the active file systems are not reflected in the active file system with which the changed active file system shares the data.</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data.</p>
<p>48. A storage system, comprising: at least one storage device; an interface to at least one computing device or network for receiving and sending information; and a controller that controls storage and retrieval of the information in the storage device, the controller operating under program control to create plural active file systems, the program control comprising the steps of: making a snapshot of a first active file system, the snapshot initially sharing data with the first active file system; and converting the snapshot to a second active file system by making the snapshot writable, with changes made to the first active file system not reflected in the second active file system, and with changes</p>	<p>Lim teaches a computer system state checkpoint mechanism. Multiple checkpoints can be taken. <i>See e.g.</i> Col. 6:45-65, 8:5-7. The checkpoints can be mounted on another virtual machine and used as an initial state, diverging over time as the virtual machine operates. <i>See e.g.</i> Col. 7: 6-18, 8:8-14. The checkpoints inherently include files. Therefore, Lim teaches plural active file systems, wherein changes made to each file tree are not reflected in the file trees with which the first tree initially shares data. A checkpoint is converted to an active file system by making it writable and using it as an initial state. <i>See e.g.</i> Col. 7: 6-18, 8:8-14.</p>

made to the second active file system not reflected in the first active file system.	
--------------------------------------------------------------------------------------	--

III. STATEMENT POINTING OUT SUBSTANTIAL NEW QUESTION OF PATENTABILITY

Since claims 1-63 of the '001 Patent are not patentable over the prior art references cited above for the reasons set forth above, a substantial new question of patentability is raised for each claim. Further, these prior art references cited above are material to the subject matter of the '001 Patent. In particular, these prior art references provide teachings not provided during the prosecution of the '001 Patent. Therefore, a substantial new question of patentability has been raised, and reexamination is respectfully requested.

CONCLUSION

Based on the above remarks, it is respectfully submitted that a substantial new question of patentability has been raised with respect to Claims 1-63 of the '001 Patent. Therefore, reexamination of Claims 1-63 is respectfully requested.

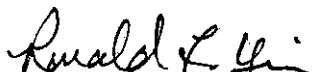
Any fee due for this reexamination may be charged to Deposit Account No. 07-1896.

Respectfully submitted,

DLA PIPER US LLP

Date: February 8, 2008

By:



Ronald L. Yin
Reg. No. 27,607

Attorneys for Requester

Ronald L. Yin
DLA Piper US LLP
2000 University Avenue
East Palo Alto, CA 94303-2248
650-833-2437 (Direct)
650-833-2000 (Main)
650-833-2001 (Facsimile)
ronald.yin@dlapiper.com